THE DESIGN AND IMPLEMENTATION OF A LOCAL AREA NETWORK CONFIGURATIONAL AUDIT TOOL

A THESIS

SUBMITTED IN PARTIAL FULFILMENT

OF THE REQUIREMENTS FOR THE DEGREE

OF

MASTER OF COMMERCE IN COMPUTER SCIENCE

AT THE

UNIVERSITY OF CANTERBURY

BY

NIGEL GRAHAM WATSON

UNIVERSITY OF CANTERBURY 1995

ABSTRACT

Configurational audit deals with identifying oversights or omissions in the use of otherwise secure computer security controls. Over the past decade, these errors, and the system vulnerability they create, have featured in many successful computer system attacks. Often, such vulnerability is easily introduced to systems, but difficult to manually detect.

This thesis deals with configurational audit tools, which are used detect such vulnerability; an overview of technical vulnerability, as well as the field of configurational audit, are provided.

A configurational audit and conformance tool called NetAudit is described. This tool, designed for the Novell NetWare 3.1x LAN operating system, uses conformance testing as its primary means of detecting vulnerability. The design and implementation of NetAudit are described, and the effectiveness of conformance testing as a means of performing configurational audit is assessed.

ACKNOWLEDGMENTS

This thesis was made possible through the generosity of the Government Communications Security Bureau postgraduate research programme. I would like to thank Ray Parker, the director of the GCSB, for his support of this programme.

Thanks to Malcolm Shore and Roy Anderson of the GCSB, for their continued support and advice throughout the project. Thanks also to Ray Hunt, my thesis supervisor here at the Department of Computer Science at Canterbury University.

Thanks are due to Hamish Marson and Brent Summers of the University of Waikato for their help with NetWare API documentation. Thanks also to John Baird of Lincoln University, NetWare programmer extraordinaire, and all-round nice guy, for his assistance with some of the finer aspects of the NetWare API.

The irrepressible John Harding of QTECH computers deserves thanks for his advice on NetWare from a hardened technician and administrator viewpoint; Nick Fitzgerald, PC support consultant here at the University of Canterbury, deserves thanks for his comments on NetWare security, and his help with lots of other things, too. Thanks also to several other anonymous NetWare administrators for their help in evaluating NetAudit in a production environment.

My brother Rick, and his wife Colleen - thanks for putting up with me, guys. Thanks to Mum and Dad, for their support, and to Gregg and Pete, of the Computer Centre, for keeping me in underwear over the period of this thesis.

Thanks to my friends Al, Tania, Kev, Kirsty, Chris, Ange, Nick & Debs for suffering (or is that enjoying) my continued absence at important social get-togethers. Thanks also to the cat, Shylock, for pointing out a number of flaws in the user interface of NetAudit.

Thanks to Richard Pascoe, for his advice on preparing a thesis, and to Hugh Emberson for help with C++ coding, as well as lots of other things. Thanks to David Bainbridge for, well, being David.

Finally, thanks to Fiona, who made this all happen.

For Fiona,

Shade of my heart.

Ninety percent of the time things turn out worse than you thought they would.
The other ten percent of the time you had no right to expect that much.

- Augustine

CONTENTS

PART II - NETAUDIT

APPENDICES

CHAPTER 1
INTRODUCTION

The last decade has seen a rapid adoption of computer system interconnection as a business tool. Local Area Networks, connecting hundreds or even thousands of organisational users, are now common business technology components. More recently, Global Area Networks, providing access to a range of useful services, and literally millions of potential customers and business partners, have experienced phenomenal growth. The competitive and strategic advantages of networked computer systems, combined with the cost-effectiveness adopting such technology, suggest that this trend can only continue.

Nevertheless, computer system interconnection is a potentially hazardous practice. In November of 1988, a rogue worm attacked and infected thousands of computer systems connected to the ARPANET [19, pp.193-221]. In June 1991, an (unidentified) Californian magazine publisher was forced to shut down its entire network of some 1500 workstations because of a widespread viral infection [43, p. 115]. In August 1992, the Texas A&M University Supercomputer Centre was the target of coordinated attacks from an unknown number of Internet attackers [65]. For the first seven months of 1994, intruders compromised an unclassified Pentagon computer system, and "stole, altered and erased records" [58]. Early in February 1995, a U.S. Internet services provider, *CapAccess*, was compromised by attackers who stole passwords for up to 12,000 users [60].

Unfortunately, such attacks are widespread. Numerous accounts of computer security problems can be found in the literature, or in media reports. For a variety of reasons, even more incidents pass unreported. Given the attention that computer security has received in the last few years, it would not be unreasonable to expect that such attacks are diminishing in frequency. However, this does not appear to be the case.

There are a variety of possible explanations: The current rapid rate of technological change means that new services appear in computer systems before the security impacts of such features have been assessed; design and implementation flaws in security controls may

render systems vulnerable to attack; and controls designed to prevent or deter attacks may not keep pace with the sophistication and resources of potential adversaries.

Computer networks present a new range of threats and potential attackers. In the past, an attacker required a certain amount of physical access to compromise a target system; networks make remote attacks possible, and as an added benefit to attackers, greatly reduce the chances of detection. Although connecting your system to a public network may mean that you can access a number of vital business services, it can also mean that an unknown number of attackers can now access you. Even within organisations, information travelling around Local Area Networks is subject to eavesdropping and modification.

Computer vulnerability has not escaped the attention of those who create operating systems. In recent times, commercial multi-user operating system vendors (such as Novell, Sun, HP, DEC and IBM), have expended considerable energy in improving the quality and strength of technical controls. Despite these efforts, successful attacks continue to occur.

Often, one need search no further than the *configurational vulnerability* of systems in order to find out why these attacks are so prevalent. Configurational vulnerability occurs when otherwise attack resistant mechanisms enforcing system security are not used, or are used incorrectly. A large number of installed systems, through simple omissions or oversights in the use of technical controls, end up vulnerable to a broad range of attacks.

This thesis focuses on tools that can help reduce the incidence of configurational vulnerability within computer systems. These tools assess the operational use of computer security mechanisms, and are able to point out flaws in system configuration that can lead to vulnerability. In later chapters, the objectives, philosophies and design of such tools are discussed.

As part of this thesis, a configurational audit tool called NetAudit was developed to address the Novell NetWare 3.1x environment.[1] This LAN operating system is commonly used by businesses of all sizes to provide PC networking functionality. As will be shown later in this document, NetWare suffers from a variety of possible weaknesses in the use of security controls, and shares many of the same security threats as other network computing environments. The planning, design and an assessment of the effectiveness of this tool are also described in subsequent chapters.

[1] In this thesis, NetWare 3.1x refers to version 3.11 and 3.12 of Novell's NetWare LAN operating system.

RESEARCH OBJECTIVES

The objectives of this research project were:

1. To review general security issues, computer security vulnerability, and the field of automated configurational audit tools.
2. To assess the NetWare 3.1x environment, with particular emphasis on possible sources of vulnerability within NetWare's security mechanisms.
3. To determine how configurational audit tools could be applied to improve the operational security of NetWare systems.
4. To develop and evaluate such a tool for the NetWare environment.

THESIS LAYOUT

This thesis is divided into a total of eleven chapters, and includes a number of supporting appendices. A chapter summary is provided below.

Chapter 2 Overviews the objectives of system security and the security environments in which system security operates. Relevant concepts from the field of Risk Analysis are introduced and explained.

Chapter 3 Examines technical vulnerability in some detail. Some general factors for the presence of such flaws are presented, followed by discussion of vulnerability and the system lifecycle.

Chapter 4 Introduces configurational audit tools. Objectives, policy issues, and hazards are discussed.

Chapter 5 Examines the driving philosophies that shape configurational audit tools. Specific tool implementation techniques are also examined.

Chapter 6 Presents a review of existing configurational audit tools. The approaches of these tools are analysed and compared.

Chapter 7 Introduces the NetAudit configurational audit tool. A discussion of why configurational audit is required for NetWare is followed by an outline of the methodology used in NetAudit's development.

Chapter 8 Explains how a set of requirements for NetAudit were developed. This chapter provides a high level overview of NetWare's security controls; a number of possible vulnerability scenarios in those controls are identified and explained.

Chapter 9 Overviews the design and implementation of NetAudit. The development of NetAudit's baselines, using vulnerability scenarios from chapter 8, are explained in detail.

Chapter 10 Describes a trial project using NetAudit. The results of this project, along with an analysis of the NetAudit approach and tool, are presented.

Chapter 11 Summarises and concludes the thesis.

The main appendices are as follows:

Appendix A Describes techniques and sources of information that can be used to discover the presence of technical vulnerability.

Appendix B Presents an in-depth analysis of vulnerability within the overall NetWare security environment.

Appendix D Presents user interface previews of the NetAudit package.

CHAPTER 2
SYSTEM SECURITY CONCEPTS

This chapter presents a contextual view of security, in which the basic objectives of System Security are explained. A formalised approach to system security, known as risk analysis, is outlined, and the major concepts of this approach are described. Finally, the various classes of information systems security are defined and discussed.

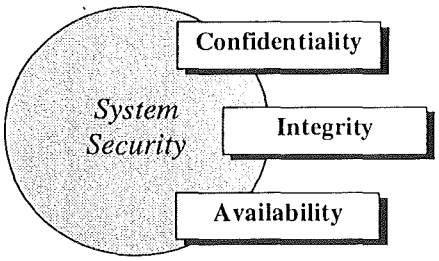


Figure 2 - 1 System Security Objectives

2.1 SECURITY OBJECTIVES

System security is the process of ensuring that the fundamental system properties of *confidentiality*, *integrity*, and *availability* are maintained to specified standards in a cost-effective manner. In order to understand the security process, it is first necessary to examine these principles in more detail.

CONFIDENTIALITY

Confidentiality exists when system components are protected from unauthorised disclosure. A system may contain information of a private nature, of commercial value, or of significance to national security. Confidentiality seeks to prevent that information from being revealed to unauthorised parties.

In the context of commercial organisations, confidentiality may simply consist of ensuring that competitors do not learn of marketing plans for the coming season. In a military context, confidentiality could mean ensuring that foreign powers do not learn of your plans to invade them within the next six months. In either case, from the point of view of those responsible for the security of the system, such disclosures are undesirable.

The strength of those measures required to achieve confidentiality for a particular system depends largely on the type of information the system stores, how attractive that information is to other interested parties, and the resources available to potential attackers.

INTEGRITY

A system can be said to posess the property of integrity if the information that it contains is both complete and correct. System integrity revolves around ensuring that sources of incoming information (data) or functionality (in the form of programs or hardware) are authentic, correct, consistant, and verifiable. System integrity can be adversely affected by deliberate or accidental modifications to any system component.

Integrity can have a direct impact on human safety. In late 1989, a hospital in Stoke-on-Trent, England, discovered that a computer programming error in a radiotherapy machine had resulted in significant underdosages being administered to patients during cancer treatment sessions. As a result, nearly a thousand cancer patients treated by the system over a period of ten years received incorrect doses of radiation. The error was only discovered after the radiotherapy system was decommissioned.(Risks-Forum [56])[1]

AVAILABILITY

A secure system must be available to provide required service and functionality to authorised parties within acceptable time constraints. This should be the case even under adverse conditions. A *denial of service* occurs when the system is no longer available and authorised users of the system are unable to acquire system resources essential for normal operation. In many ways, a denial of service can be just as damaging to a system as a loss of confidentiality or integrity.

[1] Other similar examples of medical cancer treatment systems failing due to programming errors are given in [57] and [13, p.143]

As mentioned in Chapter one, an infamous denial of service example occurred early in November 1988, when a program exploiting several common Unix vulnerabilities was released onto the ARPANET. The self-replicating program, which later became known as the Internet Worm, reduced network and system speeds to a crawl as it frantically reproduced on as many machines as it could locate.

The impact on system availability of this attack were two-fold: First, the worm caused serious and widespread system performance degradation. Second, machines infected by the worm had to be disconnected from the network and "cleaned", a time consuming and tedious process. The real cost of such a widespread denial of service was never accurately established (see Denning [19, pp.193-200, pp.260-263]).

Cost considerations

Security measures must be cost effective *relative to the value of the system components and information that they protect.* Simply put, this means that the cost of ensuring system confidentiality, integrity and availability must not exceed the cost of a failure to achieve those goals. Unfortunately, it is often difficult to assess the value of system components, and consequently the cost of a compromise of system security.

2.2 SECURITY ENVIRONMENTS

System security is not restricted simply to the examination of computer hardware, software and data. System components in a number of other environments should be considered concurrently in order to achieve overall security. These environments are summarised below:

- *The physical security environment.*

  In many cases, the most effective security measures are physical. By physically isolating a system from threats, attackers must expend more effort to gain access to system components. Components often requiring physical protection include facilities, equipment, personnel, media, and externally supplied services such as power and communications facilities.

  Physical safeguards include location, locks, fences, guards, alarms and surveillance systems. The location of backup media and facilities, the destruction of used media, and protection against natural and man-made adversaries such as flood, fire and earthquakes are also considered in the physical security environment.

- *The personnel security environment.*

  This security environment regulates which people are authorised to access system resources, and what that level of access is. Where sensitive information is processed, background checks of key personnel may be necessary. The personnel security environment addresses the question of who is required to be subjected to such tests, and how those tests should be performed.

  People may also require protection from many of the same threats that face other system components. Acts of terrorism, sabotage, and vandalism all potentially involve some danger for personnel as well as facilities (Cooper [13]).

- *The regulatory security environment.*

  Often, the continued operation of a computer system relies on its adherence to standards that are enforced and regulated by external organisations. The regulatory security environment is concerned with ensuring that the nature and operation of a system complies with any relevant regulations and laws. These include issues of copyright, privacy, correctness of information stored within the system, standards of operation, and staff conditions as outlined by the appropriate laws and regulations.

- *The hardware security environment.*

  The hardware security environment deals the computer hardware components of a system. Issues such as the integrity and reliability of hardware, access controls for devices, and controlling electromagnetic (TEMPEST) emanations from these devices, are all considered within this environment.[2]

  Safeguards within this environment include hardware redundancy (such as fault-tolerant systems), biometrics and smartcards for access controls, and electromagnetic shielding or encryption to reduce the possibility of emanation related compromises.

- *The software security environment.*

  This environment addresses threats to the operating system, applications, and utilities. Threats to these components include Trojan horses, virii, trapdoors, worms, and bugs. Safeguards in the software security environment are many and varied. For instance, they may consist of virus and change detection systems, or extensive system testing in

[2] TEMPEST is the term given to the general field of electromagnetic emanation security. According to Malcolm Shore of the NZ GCSB, the phrase TEMPEST relates "to the similarity between emitted radiation and a bad storm."

an effort to locate system bugs. Management issues, such as auditing, certification and accreditation, and configuration control, are also important safeguards.

- *The data security environment.*
  The data that resides within a system, as well as the media upon which it is stored, requires protection from unauthorised disclosure, modification, tampering, and inaccuracies. Safeguards to protect against such threats include access controls, encryption, file signatures, degaussing, and physical safeguards (in the case of media).

- *The communications security environment.*
  The communications security environment confronts threats that that affect communications between components of the system. Safeguards against communications threats include emanation (TEMPEST) controls for network cabling, encryption of data travelling on the network, and intelligent networking components that enforce network access and usage controls (such as intelligent hubs and routers, or firewall techniques).

2.3 RISK ANALYSIS CONCEPTS

To better understand the goals of system security, and how they are achieved, it is necessary to identify and define the entities involved in the security process, and the roles that those entities play. This section presents concepts from a field called *risk analysis*, a formalised approach to assessing system security that has been widely adopted by security practitioners. It is outside the scope of this thesis to discuss risk analysis methodologies in detail; more in-depth treatments of the topic can be found in Buck [6], Cooper [13], Farquhar [24], or Gilbert [27].

A useful definition of the term risk analysis is given by Gilbert [27, section 2.2]:

> *...[Risk analysis is] a procedure used to estimate the potential losses that may result from system vulnerabilities, and the damage from the occurrence of certain threats. Risk analysis identifies not only critical assets that must be protected but considers the environment in which these assets are stored and processed. The ultimate goal of risk analysis is to help in the selection of cost-effective safeguards that will reduce risks to an acceptable level.*

As stated, security risk analysis is used ultimately to select safeguards that will help eliminate or reduce the severity of risks facing the system. Risk analysis also benefits other security activities. For instance, a risk analysis project may help generate long term

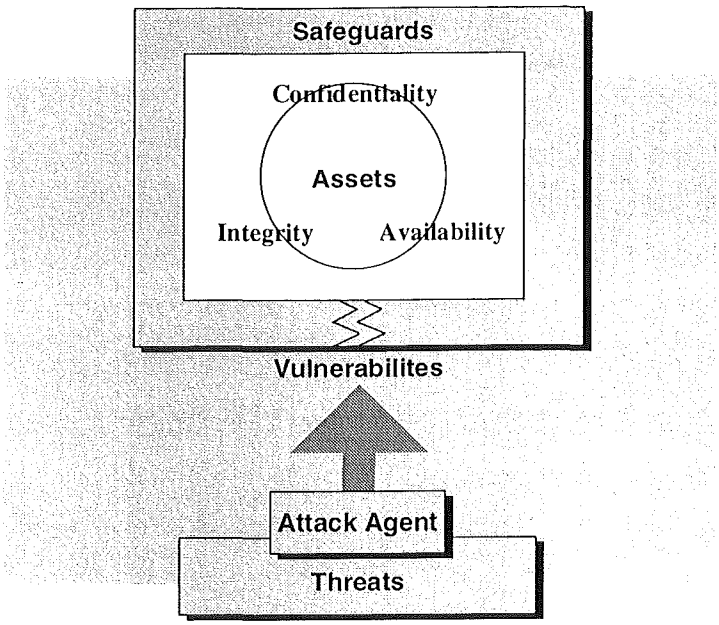


Figure 2 - 2 Generalised model of System Security.[3]

security plans and policies [13, p.27], and can also increase the general level of organisational security awareness.

Figure 2 - 2 depicts a generalised risk analysis perspective of system security. In this model, the confidentiality, integrity, and availability of system assets are protected by system safeguards. There may exist threats to those assets, which are realised by attack agents exploiting vulnerabilities in system safeguards. These concepts are summarised below.

- *Assets*

  Identification and valuation of system assets is an essential risk analysis activity. Assets are those system entities that have some value, and for which protection measures are required. Assets include tangible items such as hardware, software, personnel and facilities; Intangibles, such as goodwill, morale, reputation and opportunity are also often considered system assets.

- *Threats*

  A threat is a person, thing or event which poses some danger to an asset. The existence of a particular threat by no means implies that the threat will occur. In order to be of harm, the potential of the threat must be *realised* by an attack agent exploiting system safeguard vulnerabilities. Threats fall into one of two classes: unintentional (accidental)

[3] This diagram is an adaptation of Figure 1 of [52]

Intentional threats	Unintentional Threats
Unauthorised system access, browsing, inferencing or modifications.	Human errors.
Interception of information.	Programming errors.
Disruption of system functionality.	Electromagnetic (TEMPEST) emanations.
Theft or vandalism of system components.	Equipment failures
Misfeasance[4].	Power failures.
Social-engineering.	Communications failures.
Sabotage and Terrorism.	Changes in regulatory environments.
Acts of Commission or Omission.	Natural disasters.

Table 2 - 1 Examples of intentional and unintentional threats.

and intentional. Examples of each kind of threat are shown in Table 2 - 1. The impact of a threat occurrence is usually some function of the value of the asset that the threat affects.

- *Safeguards*

  Safeguards (or controls) are measures taken to prevent threats from occurring, or alternatively, to lessen the impact of threats that do occur [6, p.85]. These are respectively known as preventative and control safeguards. Safeguards may be embodied in policy and regulatory measures, procedural measures, environmental measures, physical measures or technical measures. A single safeguard may provide protection against a variety of threats.

  Preventative safeguards attempt to prevent a threat occurrence by removing an asset from the reaches of a particular threat entirely. For instance, the threat of a system penetration by an intruder originating from an external network may be removed by the safeguard of physically isolating that system from the network. The general aim of preventative safeguards is to elevate the cost of penetration (for the attacker) to a level that is greater than the attacker can expect to gain from the penetration. However, it is important to bear in mind that the attacker and the owner of a targeted asset may possess different ideas about the actual value of compromising that asset.

  Preventative safeguards may not totally remove the possibility of a particular threat occurring. In some cases, the safeguard may simply serve to reduce the frequency of a threat occurrence by deterring would be attackers. For example, a password protected

[4] "Misfeasance" is the deliberate abuse of authority or trust.

login mechanism may deter a majority of "casual" attackers, but will probably not discourage a more determined and resourceful attacker.

Control safeguards fall into two categories: *containment* or *recovery*. A containment safeguard limits the impact of a threat occurrence by damage control strategies, such as detecting and deflecting the attack as it is in progress, or by quickly isolating the attack target so as to limit damage to other systems. A recovery safeguard aims to facilitate system continuance in the aftermath of a threat occurrence. Backups of data, applications, and in some cases, entire sites are typical recovery safeguards.

- *Vulnerabilities*

  Vulnerabilities are weaknesses in, or a complete absence of, system safeguards. Vulnerabilities may be present in any of the physical, personnel, procedural, regulatory, hardware, software, or communications environments of a system.

  Detecting and assessing vulnerability is a major aspect of risk analysis. Without some idea of the nature of system vulnerabilities, it is difficult to assess the level of risk facing a system. Unfortunately, locating the presence of vulnerabilities, particularly in technical safeguards, is a difficult process. It is not uncommon for vulnerabilities introduced during system design to lie dormant for a number of years before being discovered and exploited. This was the case with the NetWare packet spoofing vulnerability that is discussed Appendix B.

  Even systems that incorporate particularly attack resistant technical safeguards can suffer from vulnerability. If these safeguards are not used, or are misconfigured, serious technical vulnerabilities may exist within the system. However, a false sense of security may still prevail, because the system "has the strongest protection money can buy".

- *Attack Agents*

  An attack agent (sometimes called a threat agent, or an attacker) is an entity that initiates an attack on an asset within a system. By exploiting system vulnerabilities, the attack agent causes threats to occur.

  Attack agents can take many forms. For instance, an attacker may be human, a programmed agent (i.e., a virus), or a natural agent (i.e., natural disasters). The intention of the attacker may be malicious or accidental, depending upon the nature of the threat that they are in the process of realising.

Attackers may be internal or external to an organisation. In the case of human attackers, it is generally accepted within the security community that a majority of attacks are perpetrated by internal employees [51, p.405]. Furthermore, one study revealed that up to 75 percent of security violations reviewed were perpetrated by systems administrators, with the remainder being attributed to regular users (15 percent) and outsiders (10 percent) [78].

THE RISK ANALYSIS PROCESS

Risk is typically expressed as a function of threat likelihood and the expected impact of a successful attack. Exact methods of calculating risk vary according to the particular methodology in use, but generally a risk analysis project involves the following steps (from NZSIT100 [52, p.12]):

1. Identify system assets.
2. Determine for each asset the type and potential frequency of threat occurrences.
3. Determine the impact of each potential attack.
4. Derive a risk factor for each asset, and for the system as a whole by using the impact and expected frequency of threat occurrences.

This exercise produces a risk factor for each asset, assessed in relation to the types of threats that those assets face. By selecting and applying safeguards, the level of risk for each asset, and for the system as a whole, can be reduced to an acceptable or manageable level. Because the likely impacts of successful asset attacks are (hopefully) known, safeguards can be selected on the basis of how cost-effective they are relative to the risks that each asset faces.

CONFIGURATIONAL AUDIT AND RISK ANALYSIS

Of particular importance to configurational audit is step two of the general risk analysis process above. In order for threats to occur, system safeguards must be absent or deficient. Assessing the likely frequency of threat occurrences therefore involves determining of the extent of system vulnerabilities. As mentioned on Page 28, misconfiguration of otherwise secure technical controls can lead to serious system vulnerabilities. Fortunately, safeguarding against these vulnerabilities can be very cost effective (usually involving simple reconfiguration), *as long as the vulnerabilities have been detected.*

Thus, configurational audit tools may prove to be extremely useful tools for risk analysis practitioners. First, such tools can potentially be used during a risk analysis project to

assess the level of technical system vulnerability. Second, such tools can point out cost effective means of safeguarding against a variety of technical threats.

2.4 CHAPTER SUMMARY

This chapter has introduced the main objectives of the system security process: confidentiality, integrity and availability. These fundamental principles guide the security process.

Security environments were briefly discussed. Although vulnerability may exist in any of the system security environments described, this thesis mainly concentrates on flaws that occur in the technical (hardware, software, data, and communications) environments.

Finally, concepts from risk analysis were presented. As discussed, risk analysis is concerned with assessing the level of system risk, and where possible, applying cost-effective safeguards to minimise or remove potential losses from attacks. As discussed, configurational audit tools can assist the risk analysis process.

CHAPTER 3
TECHNICAL VULNERABILITY

This chapter focuses on the notion of technical vulnerability, and the various general factors that are likely to influence the level of such vulnerability within a system. The relationship between the system lifecycle and technical vulnerability is also explored.

3.1 TECHNICAL VULNERABILITY

A technical vulnerability is *a weakness that occurs due to absences or deficiencies of controls in components of the hardware, software, data or communications environments of an information system.*

Technical vulnerability occurs as a matter of course in most computerised information systems, and only careful attention to overall security during the entire system lifecycle can minimise its presence. Nevertheless, some latent level of technical vulnerability must be accepted as an inherent part of operational systems. This chapter explores some of the reasons why this is the case.

Technical vulnerabilities generally fall into four of the vulnerability classes identified in Ruthberg [64, Appendix E]. These classes are *uncontrolled system access*, *application errors*, *operating system flaws*, and *communication system failures*.

- *Uncontrolled system access*

  Uncontrolled system access occurs when weaknesses in technical security safeguards fail to establish controls over who can access components of the system, or when those controls fail to adequately regulate legitimate system access and use. Uncontrolled system access vulnerabilities can occur in hardware, software, data, and communications services.

- *Application errors*

  Application errors may occur if an application, through design, implementation, documentation, production or maintenance flaws fails to protect the integrity, confidentiality or availability of the system.

- *Operating system flaws*

  Operating system flaws may occur in the same fashion as application errors, since operating systems are in essence another form of application. Additionally, inadequate configuration of operating system services and controls, deliberate penetrations resulting in operating system modifications, and errors during system installation may also lead to flaws.

- *Communication system failures*

  Communication system weaknesses include accidental failures, such as undetected transmission errors or accidental emanations, and failures to protect against intentional acts, such as unauthorised monitoring, compromise of intermediate network nodes, corruption of data, modification of data en-route, and unauthorised use of network services.

3.2 GENERAL CONSIDERATIONS

Although vulnerabilities may be created at any point in the system lifecycle (examined further on page 37), several general factors can influence the overall expected level of technical flaws a system contains. These factors are discussed in this section, and include:

- The size and complexity of the system.
- The distribution of components.
- Poor security practices.
- Technology gaps.
- Operating environment.
- The level of system trust.

SYSTEM SIZE AND COMPLEXITY

The size of a system can be measured in the number of components (hardware devices, software modules, files or communications subsystems) it contains. Each component may increase or decrease overall system security, and as more components are added to a system, the probability of technical vulnerability increases.

Large systems typically contain a greater number of inter-component interactions. These interactions tend to increase the overall complexity of the system. While size and complexity in themselves are not immediate problems, some difficulties may arise, as detailed below:

- *Understanding*

  Large and complex systems are less likely to be understood in their entirety by any one person. Where there are large numbers of components, it may be impossible or impractical to become familiar with the entire system. This lack of understanding, especially by those responsible for managing the system, can lead to configurational difficulties or incompatible usage problems.[1] Problems understanding large systems are compounded by inadequate documentation, which may be rendered inaccurate as system components are added and maintained as part of the normal system lifecycle.

- *Administration*

  Large systems containing many file system objects, software modules, hardware devices, and user objects, are more difficult to manage. Some production systems may contain thousands of users, and many gigabytes of file system data. Detecting configurational vulnerabilities within such systems may be difficult, simply due to the sheer number of objects and object interactions that must to be analysed. This is especially true if there are no automated tools available to assist the security evaluation process.

- *Obsfucation*

  Large and complex systems may be ideal vehicles for obscuring intentional or low visibility vulnerabilities. If no one understands how a system works, then they are much less likely to understand or identify situations where technical vulnerabilities are compromising security.

DISTRIBUTED COMPONENTS

Early models of computer security relied heavily on physical security. By establishing a secure perimeter around a central computer installation, a general approach of "trust everything within the perimeter, and nothing outside it" was feasible. This paradigm meant that less emphasis on technical safeguards was required, and more attention could be paid to physical, procedural and personnel security aspects.

[1] Incompatible usage is disussed in more detail on page 42.

In modern computing environments, the physical security perimeter approach is frequently no longer applicable. Components of the system may be located in different parts of the country, or even in different parts of the world. These components are often connected by communications systems administered by external organisations, who may or may not practise good security. Hence, an increase in the range of possible vulnerability introduction points may be observed in distributed systems.

The communications medium itself can sometimes act as an attack agent or accomplice. It may, for example, assist the proliferation of worms and viruses throughout a community of networked users. Additionally, the medium may be unreliable, or may be subject to eavesdropping by attackers.

Another source of distributed computing vulnerability stems from the proliferation of workstations and personal computers throughout organisations. Often, these devices are connected to organisational computing networks with little or no attention paid to security. Consequently, lax security habits on the part of one networked workstation user can potentially jeopardise the integrity of the entire network (Anderson [1]).

POOR SECURITY PRACTICES

The way that security is practiced by users and managers can directly impact the expected level of vulnerability, technical or otherwise, within a system. Three security practice issues affect vulnerability:

- *Security policy*

  A computer security policy is effectively a plan that details guidelines for the operation of the system. Amongst other goals, a security policy aims to prevent security problems by providing a set of guidelines that ultimately drive security procedures and practice.

  In order for a security policy to be enforceable and accepted within the organisation, it should be endorsed by senior management, and should be created with substantial input from users, administrators and management. Setting security policy will usually involve making decisions about a number of issues, as shown in Table 3 - 1.

  While other organisational policies may make statements about required design, implementation, testing, documentation and maintenance standards, a lack of day-to-day computer security policies may lead directly to vulnerabilities. For example, an organisation may lack policies regarding the selection of secure passwords. As a result,

Policy Issue
1. Who is allowed to use system resources
2. What is the proper use of resources.
3. Who is authorised to grant access and approve usage.
4. Who is allowed system administration privileges.
5. What are the user's rights and responsibilities.
6. What are the system administrators rights and responsibilities of the system administrator as opposed to those of the user.
7. Who is responsible for auditing and reviewing system security.
8. What happens when policy is violated.

Table 3 - 1 Computer security policy decisions (adapted from [32, p.13]).

users may end up using short passwords, passwords that are some permutation of their login name, or passwords that are locatable in standard dictionaries.

- *Administration issues*

  System administration practices can influence the presence of vulnerability. Where possible, administration duties should include monitoring system use, reviewing system security, educating users about secure computing habits, and monitoring currently known system vulnerabilities. The level of skills possessed by the system administrator also directly influences system vulnerability: inexperienced or unskilled administrators are more likely to make mistakes that result in security mishaps.

- *User habits*

  The activities of individual system users can potentially increase or decrease system vulnerability. For example, users who transfer confidential files across untrusted public networks, or who create world readable and writable files, increase the overall level of system vulnerability.

  One possible approach to user created vulnerability is to educate users about appropriate system habits and standards (see previous point). Another approach, examined in this thesis, is to periodically audit user habits and system objects, looking for potential areas of vulnerability.

TECHNOLOGY GAP

Neumann and Parker [51] define the technology gap as:

> ... *the [technological] gap that exists between what a computer is actually* capable of enforcing *and what it is* expected to enforce...

Level	Featuring
D	Minimal Protection
C1, C2	Discretionary protection
B1, B2, B3	Mandatory protection
A1	Verified protection

Table 3 - 2 Orange Book security evaluation classes.

A computer system may be expected to enforce certain policies regarding system confidentiality, integrity and availability, but due to deficiencies in its technical environment, may not be able to do so. For example (from [51]), discretionary access controls such as user/group/world protections are intended to limit file system access, but are incapable of enforcing copy protection. Such gaps create vulnerabilities within the system that could be exploited by an attacker.

OPERATING ENVIRONMENT

Systems operating in a benign environment are less likely to be attacked. On the other hand, if the environment is hostile, attacks may be more common. Note, however, that security standards in a benign environment may be lower than those of a system in a hostile environment; therefore it is likely that systems existing in a benign environment suffer from an increased probability of a *successful* attack.

THE LEVEL OF SYSTEM TRUST

The final general factor that influences the likelihood of technical vulnerabilities is the level of trust to which the system has been assured. The United States government's *Trusted Computer System Evaluation Criteria* (the "Orange Book" or TCSEC) [20] defines four broad divisions of security protection and trust, as shown in Table 3 - 2:

Each evaluation level, from D through to A1, features an increase in the amount of trust that is inherent in the system. At the lower levels of the classification (C1, C2 and B1), assurance of correct and complete security is mostly provided by extensive testing of security features. At the higher levels (B2, B3), assurance becomes a design and implementation consideration, while at the highest level (A1), the design of the system is formally specified and verified.

Technical vulnerability is less likely to exist in systems that have been certified at a level higher than B1, and is almost certain to be rare in systems certified at A1. Notwithstanding this, systems that have been evaluated to TCSEC standards may still contain vulnerabilities. This certification simply means that users of the evaluated system have more assurance that

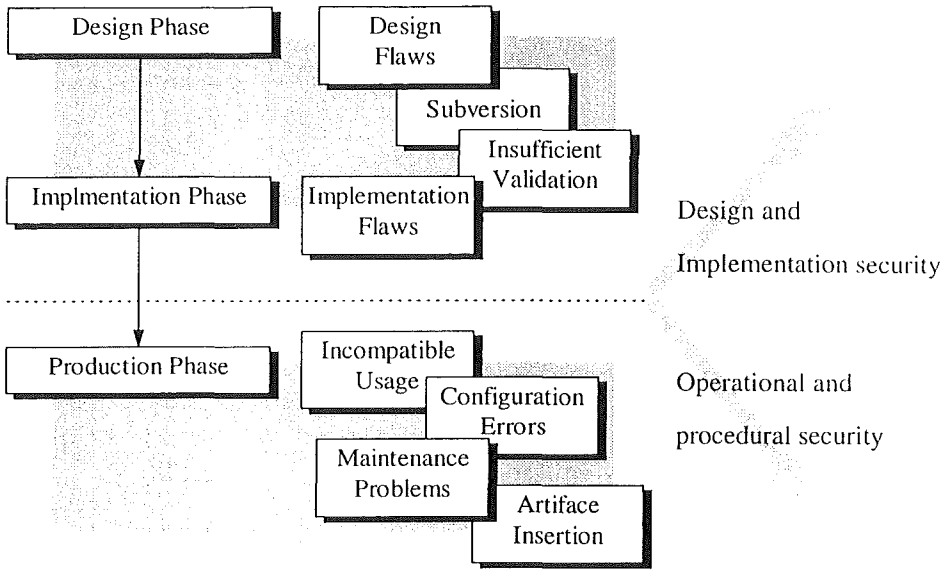


Figure 3 - 1 Vulnerability introduction during the system lifecycle.

those vulnerabilities that are still present require inordinate amounts of effort on the part of the attacker to exploit.

TCSEC certifications are no assurance, especially in systems lacking mandatory access controls, that the usage of security mechanisms will be consistent with organisational security requirements. Even when mechanisms that implement technical security policies are extremely resistant to attack, improper or incorrect use of those mechanisms can still lead to weak security. These configurational vulnerabilities are discussed further on page 43.

Other classification schemes exist besides the TCSEC model. For example, the Harmonised ITSEC criteria [31], based on TCSEC and a number of European security classification documents, emerged in 1990. This criteria has seven increasingly stringent classes of assurance (E0-E6), and has been adopted by the New Zealand government as a national standard for evaluating government computer systems and products (Farquhar [24, p.111]).[2]

3.3 THE SYSTEM LIFECYCLE

Besides the general considerations introduced in the previous section, technical weaknesses in system components may be introduced at any point in the system lifecycle. The impact

[2] For a number of reasons, the TCSEC criteria are considered mostly obsolete. They are due to be replaced by the new Federal Criteria for Information Technology Security [25], which address a wider range of security issues.

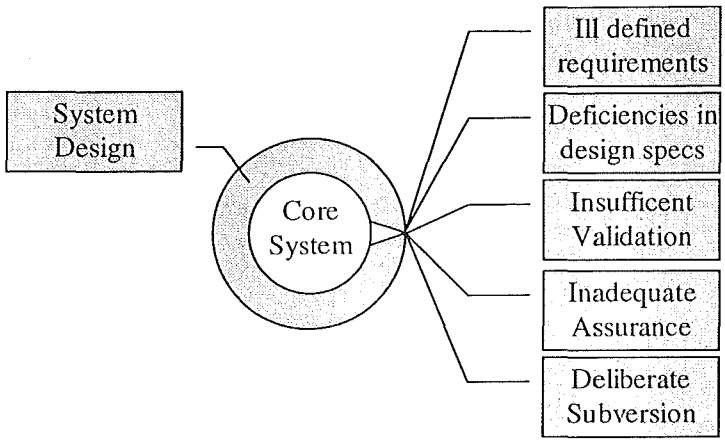


Figure 3 - 2 System design vulnerabilities.

and remedy of a technical weakness may be largely influenced by the lifecycle stage in which the vulnerability was introduced. The earlier in the system lifecycle a flaw is introduced, the greater the potential havoc that flaw is capable of generating, and the more expensive it will be to remedy.

Some researchers (i.e. Myers [44]), as well as the TCSEC and ITSEC security criteria mentioned in the previous section, point to the need for attention to vulnerability to be given at every stage of the system's life - not just during the operational phase. This section focuses on the various stages of the system lifecycle, and summarises the nature of possible vulnerabilities that may be introduced at each of these stages. As shown in Figure 3 - 1, the introduction of technical flaws may occur in any of three distinct phases: the *system design* phase, the *implementation* phase, and the *production* phase.

SYSTEM DESIGN

The requirements and design phases of a system are crucial times in the system lifecycle. During this phase, key decisions regarding hardware and software specifications are made. Undetected vulnerabilities occurring at this phase of the system's life may prove impossible or inordinately expensive to safeguard once the system reaches the production stage.

Vulnerabilities at the design stage, as shown in Figure 3 - 2, include:

- Ill-defined requirements,
- Deficiencies in design specifications,
- Insufficient design validation,
- Inadequate quality assurance, and
- Deliberate design subversion.

As has been long understood by software reliability advocates, discovering and remedying problems in the pre-implementation stages of a system's life tends to be more economical in terms both of resources and time than attempting fix those same errors once the system has been implemented or is in production. One researcher estimated that fixing errors in the pre-implementation stages of system development costs about 1 per cent as much as fixing code once it has been implemented (Pfleeger [54, p.56]). In addition, the TCSEC and ITSEC standards (discussed earlier in this chapter) both make assertions about the security implications of decisions made at the design stage.

Conceptual design errors or omissions are particularly dangerous, since unremedied flaws in system design usually become embodied in the implementation of a system. Safeguarding against vulnerabilities inherent in the system design is subject to three possible difficulties:

- *Fundamental design limitations.*
  It may prove impossible to adequately "work around" a vulnerability. For instance, the MS-DOS operating system contains fundamental design limitations that prevent it from providing a secure computing environment. MS-DOS lacks file system access controls, file encryption mechanisms, memory protection, and a means to audit user activity. Safeguarding against some of these vulnerabilities can be achieved with additional (third-party) controls.[3] Nevertheless, some inherent limitations may still remain. For example, the design of DOS precludes the addition of memory protection safeguards.
- *Dangers of retrospective safeguarding.*
  Retrospectively adding safeguards may introduce new vulnerabilities. A system is usually designed with a particular set of objectives in mind. Adding to or changing those objectives may invalidate some of the original assumptions that were made during the initial design process. Dependencies between system components increase the risk that new vulnerabilities may arise from such changes due to unforseen interactions. In addition, if the safeguards involve additional coding, then this coding is subject to implementation introduced vulnerabilities (discussed below).

[3] For example, packages described in [16] (Cental Federal Systems Net/Assure) and [17] (Pyramid Development Corp Net/DACS) add improved access control, inegrity, auditing and authentication measures to MS-DOS.

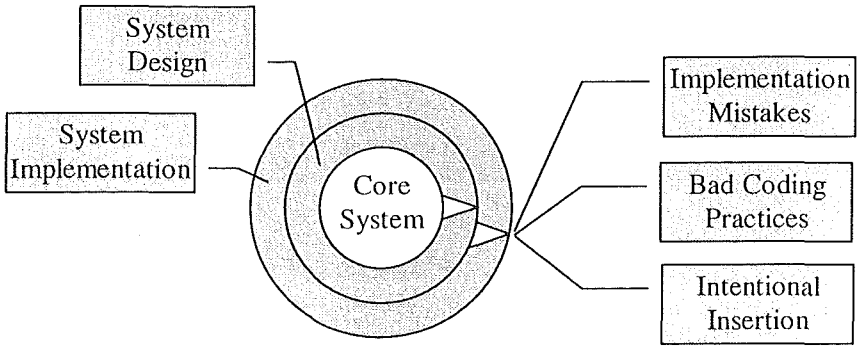


Figure 3 - 3 System implementation vulnerabilities.

- *Prohibitive costs.*

  The costs, relative to the value of assets involved, of implementing changes to an original design can sometimes be so prohibitive that it effectively becomes cheaper to scrap the original system and redesign from scratch.

SYSTEM IMPLEMENTATION

Once the system design has been completed and validated, the next major stage in the life of a system is implementation. During this phase, the design of the system is embodied as a working product. This includes the manufacture of hardware components, and the coding of software components. As Figure 3 - 3 illustrates, flaws may be introduced at this stage due to:

- *Mistakes*

  Simple mistakes during component implementation are a common source of vulnerability. Recently, Intel Corporation's Pentium CPU shipped with a serious floating point division flaw.[4] This flaw, which can potentially cause erroneous results to be encountered in high precision calculations, was the result of a simple error copying an internal table during chip manufacture. The mistake was missed during pre-release quality control checks, and eventually became a source of major embarrassment and expense for Intel.

- *Bad coding or manufacturing practices*

  Vulnerabilities introduced in during the implementation of a design can sometimes be traced to a lack of coding standards and guidelines. There may be weaknesses in the quality control and testing environments or methodologies, and system implementation documentation may be inadequate or non-existent.

[4] See [59] for a discussion of this flaw.

For example, the Unix `finger` flaw, exploited by the Morris worm, exploited a buffer overrun bug in some versions of the Unix fingerd daemon. By providing more data to the daemon than its receive buffer had space for, the worm was able to rewrite portions of the fingerd internal program stack, altering its behaviour for its own purposes [61]. This flaw existed because certain portions of the finger daemon code lacked buffer bounds checking - an example of bad coding practice on the part of the implementors.

- *Intentional insertion*

  Vulnerabilities may be deliberately introduced into the system by implementors, either overtly, in order to aid the system testing process, or subversively, in order to set up some future attack. For example, during implementation and testing, an operating system programmer might insert a "convenience" trap-door that circumvents normal access control and authentication mechanisms. The intention may be that the trap-door remains in place until just before the system reaches the operational stage. Unfortunately, because of oversights or errors, the trap-door may not be removed, and may escape into production systems.

  More malicious vulnerabilities may be intentionally inserted. For example, a programmer may insert a destructive logic bomb into an in-house payroll application that is triggered when that programmer's name can no longer be found on the company payroll.

  Ken Thompson, in his ACM Turing Award Lecture *Reflections on Trusting Trust* [76], describes an intentionally inserted vulnerability in the original Unix `login` program. This vulnerability, which involved modifications to the Unix login program and C compiler, allowed Thompson to login into Unix machines with a special master password encoded as part of the login executable. The modifications to the compiler ensured that the vulnerability would persist between recompiles of both the login program itself, and of the C compiler that inserted the vulnerability. To complete the deception, Thompson took steps to ensure that no traces of the modifications were left in the source code.

These vulnerabilities are very hard to detect, especially if they are carried out in a deliberately subversive manner. At the conclusion of [76], Thompson notes:

> *The moral is obvious. You can't trust code that you did not totally create yourself (Especially code from companies that employ people like me). No*

amount of source-level verification or scrutiny will protect you from using untrusted code.

Design and implementation vulnerabilities have implications for the operational use of systems. If such a vulnerability is found after the system has reached the production stage, then measures may be required to reduce or remove the possibility that such vulnerabilities are subsequently exploited by attackers.

One of the roles of security audits during system production is to identify systems that contain (acknowledged) design defects, and assessing whether measures have been taken to correct or ameliorate the risks presented by those defects. For instance, in operating systems, such defects may be addressed by releasing patches for the affected components. Assessing the security of that system may involve determining whether or not such patches have been installed.

SYSTEM PRODUCTION

The production phase of the system begins when the system is installed and becomes operational. Once the system reaches this stage, the vulnerability emphasis shifts from design and implementation methodology issues to operational and procedural security issues. As shown in Figure 3 - 4 on page 43, several factors could lead to vulnerabilities within operational systems:

- *Incompatible Usage*

  Unforseen interactions occur when a system is assembled or configured in such a way that components interact in unexpected or undesirable ways. These vulnerabilities, referred to by Muffett [41] as "incompatible usage" flaws, are usually accidental in nature, and are often low visibility. For example, certain incompatible combinations of hardware and software may lead to random system crashes.

  The level of experience of the system managers can be an important factor in reducing the number of vulnerabilities that are introduced through unforseen interactions. The more experienced the assemblers and maintainers of the system, the less these will occur.

  As mentioned in the previous section, the size and complexity of a system can also influence the number of unforseen interactions within a system.

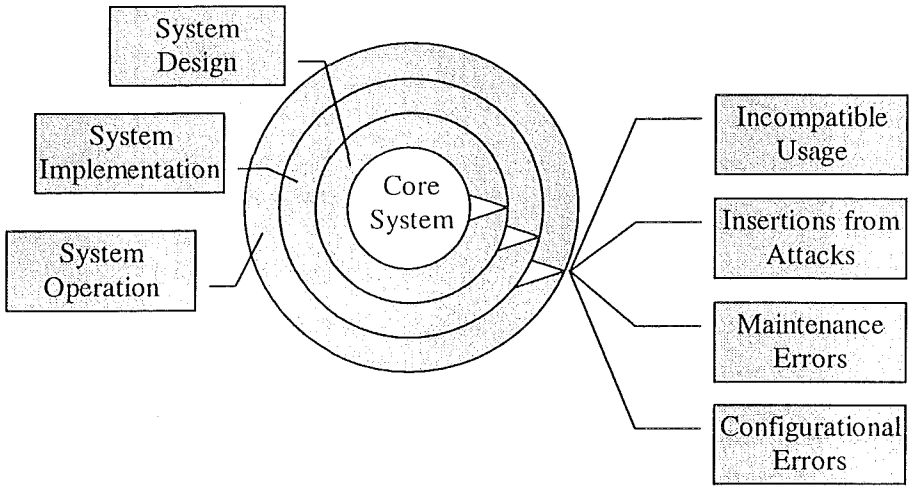


Figure 3 - 4 System production vulnerabilities.

- *Inserted vulnerability (Artifice insertion)*

  Once the system is operational, attackers may exploit existing vulnerabilities in order to insert further flaws into hardware, software, data or communications components. This insertion may be perpetrated by a programmed attacker (for instance, a worm or viral attacker), or a human attacker (for instance, an administrator making subversive changes to system data).

- *Maintenance Errors*

  Normal system maintenance can lead to a possible increase in technical vulnerability. If maintenance activity involves writing new code, or changing existing code, then implementation errors may be introduced. On the other hand, if maintenance involves reconfiguring hardware or software components, there is an increased risk of incompatible usage or configurational vulnerabilities being introduced.

  Other types of vulnerability may be introduced during maintenance activity. For example, system documentation could become outdated, or an attacker could take advantage of lapses in normal production security, and be able to insert vulnerability artifices.

- *Configurational Errors*

  Configurational errors occur when the existing safeguards are incorrectly or incompletely set up. Configurational vulnerability is discussed in more detail in the next section.

3.4 CONFIGURATIONAL VULNERABILITY

Most modern system safeguards, when used correctly, offer reasonably strong security. However, when those same controls are incorrectly used, or are not used at all, vulnerabilities will occur regardless of the strength of underlying mechanisms. Alternatively, the controls themselves may be flawed, but those flaws may have acknowledged and readily available safeguards.

From these facts, a definition of technical configuration vulnerability can be derived:

> *Configurational vulnerability is a technical system weakness brought about by the incorrect or incomplete application of existing system control mechanisms, or due to known, but unremedied, flaws in those technical controls.*

Flaws in configuration can account for a large percentage of the typical vulnerabilities exploited by attackers (numerous examples can be found in Stoll [75], [19], and Garfinkle *et al.* [26], amongst others). This type of vulnerability includes errors in the configuration of hardware devices, system services, user accounts, file system permissions, and network subsystems.

The presence of configurational vulnerability is influenced by most of the general factors discussed in section 3.2, and many of the lifecycle factors described in section 3.3. Vendor supplied defaults are also a rich source of configurational vulnerability. Farmer and Spafford [23] comment that:

> *... Even machines that have come straight from the vendor are not immune from [procedural] security problems. Critical files and directories are often left world-writable, and configuration files are shipped so that any other machine hooked up to the same network can compromise the system.*

As an example of configurational vulnerability, consider the following potential login mechanism vulnerabilities:

- The mechanism is turned off.
- The system has several non-disabled default account/password combinations.
- Some user accounts do not have passwords, or have weak passwords.
- The system supports encryption of passwords transmitted across networks, but these features are disabled.

- The login mechanism has a known bug that bypasses the authentication process when certain keystroke combinations are encountered.

Configurational vulnerabilities are a central theme of this thesis. Tools for assessing the presence of these flaws are discussed in subsequent chapters.

3.5 CHAPTER SUMMARY

This chapter has looked at the notion of technical vulnerability in some detail. General factors influencing the level of technical vulnerabilities within a system were described, followed by an analysis of how technical vulnerabilities may be introduced at various stages of a system's lifecycle. Appendix A examines some general methods that are used to locate vulnerabilities within systems, while the next chapter introduces automated tools for detecting the presence of configurational vulnerabilities.

CHAPTER 4
AUTOMATED CONFIGURATIONAL AUDITING

This chapter introduces the concept of automated configurational auditing. The objectives of a configurational audit are discussed, followed by an examination of some specific policy issues that affect the deployment, use and implementation of configurational audit tools. Some potential hazards of configurational audit tool analysis are also discussed.

4.1 CONFIGURATIONAL AUDIT

A *configurational audit* is an exercise that is undertaken in an attempt to locate configurational vulnerability. Due to the size and complexity of many systems, various tools have appeared in the last few years that help to automatically search systems for these weaknesses. This part of the thesis discusses configurational audit tools in more detail, and is divided into three chapters as follows:

Chapter 4 Focuses on objectives, policies and hazards of configurational audit tools,

Chapter 5 Discusses how configurational audit tools are implemented, and

Chapter 6 Examines some existing configurational audit tools.

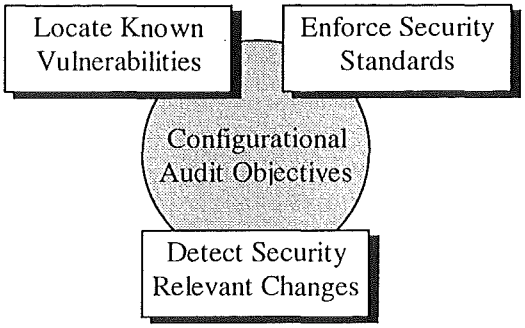


Figure 4 - 1 Configurational Audit objectives

TOOL OBJECTIVES

There are three main objectives that automated configurational audit tools may strive to attain. These objectives are summarised in Figure 4 - 1, and are discussed below.

- *Locate known vulnerabilities*

  The most common configurational audit objective is to locate known vulnerabilities. These vulnerabilities may consist of weaknesses in the configuration of otherwise secure safeguards, non-use of safeguards, or installed but flawed safeguards. Audit tools can additionally help to detect vulnerabilities that have been inserted into the system during a system attack.

- *Enforce standards*

  Configurational audit tools may check system conformance to organisational security standards dictating accepted levels of security. "Accepted" standards often vary between organisations and individual sites; for instance, academic sites have different security requirements when compared to military or commercial sites.

  Configurational audit tools, in conjunction with appropriate security policies, could be used as a device to assist the certification and accreditation of operational systems. Such tools can also be used to assess systems before they are used to process sensitive information, or before they are allowed to connect to potentially hostile networks.

- *Detect security relevant system changes*

  A third possible goal of configurational audit tools is to assess the integrity of vital system objects, by detecting changes in object security state. Some security problems, such as viruses, worms, or other types of inserted or programmed vulnerability, may be detected as a change in the state of some system object (such as a binary system file). Alternatively, changes in the state of other system objects, such as a sudden elevation of privilege for a previously unprivileged user, may also indicate that a successful attack has occurred.

4.2 POLICY ISSUES

Before configurational analysis tools are deployed within an organisation, some basic policy decisions must first be addressed. While configurational audit tools are quite often suitable for ad-hoc use, regular and systematic audits require planning and forethought. Some specific policy decisions are discussed in this section, including:

- Which personnel are involved in configurational audits.
- When tests should be carried out.
- The scope of configurational audit.
- The depth of system testing.
- What to do about detected vulnerabilities.

WHO TO INVOLVE

There is a need for clear policy guidelines regarding which personnel are responsible for the various operational aspects of configurational audit tools. Personnel factors that should be addressed are:

- Who is responsible for installing and configuring tools.
- Who is responsible for scheduling and running tool analysis sessions.
- Who is responsible for interpreting results and actioning changes required as a result of tool analysis sessions.

The answers to these questions will depend very much on individual site security requirements. In some organisations, all three of these functions will be performed by the system administrator. In others, an independent auditor may be responsible for instigating and performing configurational audits.

WHEN TO TEST

In order to provide effective coverage, configurational audit tools should be used on a regular basis. On most systems, the mix of system objects such as users, files and devices is constantly changing. As a result, access controls that protect those objects may not keep pace with this process of ongoing change.

Audits periodically test those controls to ensure that adequate security is being maintained. Depending on the site, monthly checks may suffice. However, at sites where many configurational changes occur, weekly or even daily checks may be necessary.

Audits should be carried out when the system is first installed, to counter the possibility of weak installation defaults. Configurational analysis may also be performed as part of a risk

Situation	Reason(s)
After a change in security policy.	A change in site security policy may mean that existing system controls are no longer adequate. Configurational audits assist detecting system objects that do not comply with the new policy.
After major security state changes	Major alterations in the state of system security may occur because of additions, deletions, or changes to system components such as: files, users, groups, applications, operating system binaries, network services, hardware devices, and authentication systems. After these changes, checks may be required to ensure that the system is at least as secure as before the change occurred.
After an attack has occurred.	The security state of a machine is always uncertain immediately following an attack. Doubts may exist about what (if any) changes have been made to system controls. This is especially true if attackers acquired privileged accounts during the attack. A configurational audit will help assess where possible subversive changes have been made to the system by the attacker. Change detection systems may help locate binary, configuration, or data files that have been altered during the attack.

Table 4 - 1 When to perform an "unscheduled" configurational audit (from [55, 5.1]).

analysis exercise (to assess the level of system vulnerability). There are three other extraordinary situations, summarised in Table 4 - 1, which may warrant additional configurational security checks.

TESTING SCOPE

The scope of configurational audit activity needs to be considered as a policy issue. For example, if audits encompass factors external to the technical control environment (such as assessing physical workstation security), then additional manual information gathering methods will need to be devised. In this case, checklists may be used to record whether workstation locations are physically secure when authorised personnel are not present [55, 5.1].

The scope and extent of testing within the technical arena should also be considered. For instance, policy should determine if isolated components of the system (such as user configuration files, or system binaries) are assessed, or alternatively if the vulnerability of the entire system is considered instead. This decision may again depend on individual site requirements.

TESTING DEPTH

Some systems may be more critical to organisational business continuance or security than others. Consequently, these systems sometimes require a greater depth of analysis than less "important" systems.

For example, systems that monitor or protect organisational computer security from external (network) based threats, such as firewall machines[1], require a greater depth of checking than other "non-critical" systems inside the organisational network perimeter. Since the integrity of the internal network is linked to the integrity of the firewall, the firewall system is an excellent candidate for deeper checking than "normal" machines inside the network.

MANAGING DISCOVERED VULNERABILITY

Once the configurational audit process has discovered a vulnerability within a system, the question of what to do with this information is posed. Two strategies for dealing with detected configurational vulnerabilities are available:

- *Ignore the vulnerability and manage the associated risk.*
  The configurational auditor may decide that the discovered flaw does not present enough of a risk to the system to warrant attention, and that the probability of an exploitation of the flaw, or the impact of a successful attack, is minimal.

  Alternatively, the vulnerability may be an unavoidable side effect of a particular security setup. For example, users requiring access to files while using a particular application on a LAN may also have access to those same files while not running the application. While this situation may not be desirable, the security mechanisms of the system may not support any other alternative.

- *Act upon vulnerability information.*
  In other cases, the auditor may decide that the discovered vulnerability is serious enough to warrant immediate attention. Of special interest are vulnerabilities that when exploited could compromise the entire system, or where the impact of the exploitation is unacceptable.

Some configurational audit tools may offer active repair features. Decisions regarding how these features are applied may also need to be made as a matter of policy. This is discussed further in *Active vs. passive tools* on Page 59. Assessing the magnitude of a vulnerability may be of some assistance in deciding how that vulnerability should be addressed. Vulnerability magnitude is discussed in *Reporting policies* on Page 62.

[1] A firewall system is a machine that protects internal networks from external threats, such as hackers and worms. Firewalls may implement policy regarding which external sites are allowed to access internal machines and vice-versa. In addition, firewall systems may disallow certain types of incoming connections, such as ftp or telnet services.

4.3 POTENTIAL TOOL HAZARDS

Configurational audit tools are not without hazard. Two main problems should be considered by tool developers and users: tool misuse, and tool subversion. These issues are discussed in this section.

TOOL MISUSE

There is a possibility that configurational audit tools may be used by an attacker to compromise the systems they were designed to assess. Running such a tool may reveal to a skilled attacker a multitude of potential attack avenues.

Tools distributed in source form could also be modified by attackers for their own purposes. For example, an active checker may initially only check for the presence of a Unix system access vulnerability. An attacker may modify such a tool so that if it notices that the flaw is present, it automatically takes advantage of the weakness, and retrieves the system password file. This type of attack is possible with the ISS [37] tool.

Tool misuse problems can be managed in four main ways:

- *Ignore the problem.*
  The most simple approach to the problem of tool misuse is to ignore it. In this case, a conscious (or unconscious) decision is made that the risks of configurational audit tools being exploited by attackers to streamline the attack process are acceptable. This decision may be mitigated somewhat if the tool presents little risk to security. For instance, change detection tools are unlikely to be used during attacks, and hence making them publicly available presents little risk.

- *Provide as little information about vulnerabilities as possible.*
  The second approach is to assume that the tool will fall into the hands of potential attackers. In this case, a damage control exercise is undertaken to ensure that potential attackers are not able to use the tool to learn of vulnerabilities of which they were not already aware. Instead of providing a comprehensive explanation of the vulnerabilities that have been found, such a tool gives terse messages explaining simply that a particular vulnerability exists in the system. From that starting point, it is up to the user of the tool to determine the nature and seriousness of the vulnerability.

  The problem with this "closed lips" approach is that attackers often have a better idea of system vulnerabilities than do system administrators. The administrator is placed at a disadvantage because he knows that a vulnerability exists, but is not sure how serious

that vulnerability is. Thus, the administrator is placed in a position where he or she does not know how urgently a flaw needs to be fixed, or even how to go about fixing it.

The attacker, on the other hand, may have prior knowledge of the detected vulnerability, and may be able to immediately exploit it. Alternatively, the attacker may have more time and resources than the system administrator to assess the potential exploitability of the vulnerability.

- *Secure the tool itself.*
  The tool itself may implement safeguards against possible misuse. For example, a configurational audit tool may only allow itself to be executed by a privileged user, such as the system administrator or auditor.

  This measure may not be feasible where the tool is distributed in source code or script form, as such access controls may be removed by an attacker. Additionally, attackers may use the source code to learn about system vulnerabilities.

  Securing the tool itself also limits its use by non-privileged users. For example, a site may want to actively encourage users employ such tools to check the security of their personal configuration. Limiting the tool so that only privileged users are able to use it may preclude this approach.

- *Secure the distribution of tools*
  The last approach to the problem of tool misuse is to secure the distribution process. This is generally achieved by distributing tools only to trusted parties, and carefully tracking copies of the tool to ensure that only authorised parties receive it.

  As with all distribution arrangements, this method is only as secure as the people who take part. There is always the danger that if one copy of the tool "escapes" the secure distribution process into the hands of a malicious user, then that tool will quickly be promulgated throughout the computing underground. This may occur if one of the trusted parties on the distribution list is less than trustworthy, or if an attacker steals the tool from an unsecured site.

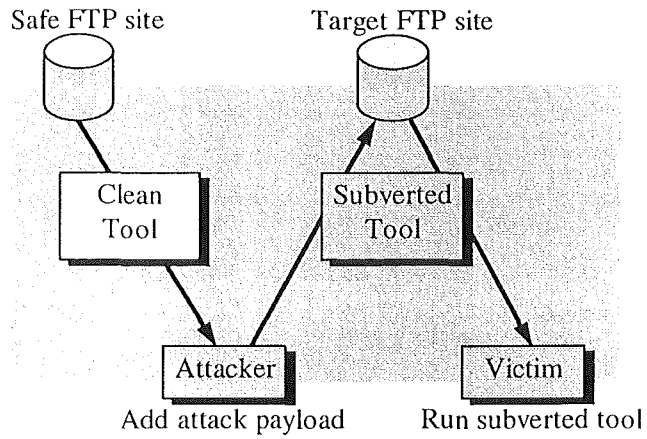


Figure 4 - 2 Tool Subversion attack scenario.

TOOL SUBVERSION

Where configurational analysis tools are publicly distributed, a risk exists that the tool may be subverted by an attacker. In an example scenario, shown in Figure 4 - 2, an attacker acquires a "clean" version of the tool from public archives, and inserts a Trojan horse, trap door or virus payload. The attacker then takes measures to obscure this addition, and redistributes the modified tool (perhaps at a different site). The victim subsequently downloads the tool from the target site and runs it, whereupon the subverted tool performs its attack with the victim's privileges.

This approach may prove inordinately successful for the attacker, since configurational audit tools are often run with elevated system privileges. In addition, installations may not have good site polices regarding installation of publicly acquired utilities, and may not check the tool for programmed threats before it is run. Three guidelines may reduce the risk of these attacks.

- *Acquire configurational audit tools from trusted sources.*
  Configurational audit tools (or indeed any publicly available tools) should only be acquired from trusted sources. For example, COPS is available on many ftp servers around the Internet. However, the best place to acquire it is from a ftp site that is known to be safe, such as `cert.org` (the Computer Emergency Response Team site).

  Where possible, tools should be distributed with secure methods. For example, public key/private key authentication and encryption schemes could be used to reduce the risk of subversion. Digital signatures are now a common form of authenticating publically distributed files, and so where possible, these should be used during tool distribution.

- *If possible, check the source before running the tool.*
  Where the tool is distributed as a binary only, this measure is not available. However, where the tool is distributed in source form, installers of the software should take the time to become familiar with the components that comprise the tool.

- *Where possible, do not run configurational audit tools with elevated privileges.*
  Very few configurational audit tools need to run with super-user privileges, and hence this is a practice that should be avoided wherever possible. Possible exceptions may be where the tool requires complete access to the file system (such as during file-system searches).

4.4 CHAPTER SUMMARY

This chapter has introduced configurational audit tools. Three tool objectives were identified: locate known vulnerabilities, enforce security standards, and detect security relevant changes. Policy issues surrounding the operational use of configurational audit tools were introduced, and some potential hazards of such tools were discussed. The next chapter delves further into configurational audit tool design and implementation issues.

CHAPTER 5
AN OVERVIEW OF CONFIGURATIONAL AUDIT TECHNIQUES

This chapter examines configurational audit tool design philosophies and describes some of the implementation techniques commonly used by publicly available tools.

5.1 INTRODUCTION

As discussed in previous chapters, configurational audit consists of analysing operational systems for known or recognisable flaws in existing technical controls. The techniques used to detect these configurational vulnerabilities usually consist of relatively straight-forward tests. However, some underlying design philosophies influence the final form of tools, and the checks they perform. This chapter first explores those philosophies, and then considers a number of implementation techniques that are commonly employed by configurational audit tools.[1]

5.2 PHILOSOPHICAL APPROACHES

The philosophical approach a configurational audit tool adopts in detecting vulnerabilities tends to have a major impact on the implementation and usage of the tool. General approach issues are described in this section, and include:

- Security areas to check.
- General or platform or system specific checks.
- Passive or active testing.
- Depth of analysis.
- Distribution of testing.
- Reporting policies

[1] COPS, TAMU ISS and TripWire are reviewed in Chapter 6.

Category	Typical Checks Performed
Access control strength and configuration	• Assess strength of user account passwords. • Check general file system permissions (especially those protecting system configuration files and binaries). • Ensure network service authentication mechanisms are in place and correctly configured. Ensure that trusted systems really can be trusted.
User accounts	• Ensure user configuration files are protected against unwanted modifications. • Check contents of user configuration files for insecure settings (i.e. insecure PATH statements, risky default file permission masks, untrusted network hosts in .rhosts files, etc.). • Assess access permissions on user-owned files.
File system	• Check that appropriate file system protections are effective for application and system binaries, and that only authorised users can access sensitive system data. • Detect unanticipated changes to system binaries. • Search for questionable file system objects, such as hidden files or directories, objects with strange names, and security relevant files (such as Unix SUID scripts).
System services	• Assess configuration files for system services (especially those that run at elevated privileges) for appropriate settings. • Ensure that configuration files are protected from unauthorised modifications. • Check that latest bug free system binaries are installed. Check for presence of binaries with known flaws.
Network services	• Ensure network accessible public areas of the file system have appropriate file protections in place. • Ensure that only authorised clients can connect to network services. • Check that debug mode type vulnerabilities are removed.
Auditing and logging services	• Ensure log files are protected from unauthorised modification. • Check that logging programs are active.
Applications	• Ensure application installation programs do not select vulnerable defaults. • Check file access permissions of applications. Check "out-of-application user access". • Check ability of captive applications to "shell" out to operating system.

Table 5 - 1 Summary of candidate check areas for configurational audit tools, and some typical checks performed in each area.

CANDIDATE CHECK AREAS

Configurational audit tools check a variety of system areas. Some tools are more specialised than others, and may only check a single isolated vulnerability area, while others may check for a range of diverse vulnerabilities. The general areas that are checked by automated configurational audit tools, and the types of checks that are performed in each area, are summarised in Table 5 - 1 on page 58.

Depending on the target platform, there may be substantial crossovers between checking categories. For example, some file objects, such as configuration files, require both protection from unauthorised modification, as well as a check of the file contents to ensure the settings they specify are secure.[2]

GENERAL OR PLATFORM SPECIFIC

A configurational audit package may be implemented with a specific target platform in mind (i.e. a brand and version of the Unix operating system), or it could be designed to work with a wide range of platforms (i.e. any version of Unix). Tools that are designed to analyse a specific platform are generally able to perform more comprehensive checking for vulnerabilities that are peculiar to that particular system. Unfortunately, such tools are potentially difficult to port to other platforms, due to system specific nature of the tests performed.

A workable compromise (adopted by both COPS [23] and TAMU [65]) is to base the tool around core generic tests, and then build in support for system specific tests. For instance, TAMU tests for generic Unix configurational vulnerabilities, but has additional system-centric testing modules which test for specific vendor/version flaws in system binaries.

ACTIVE VS. PASSIVE TOOLS

As noted by Polk [55], configurational audit tools can adopt either a passive or active approach to system testing. Passive configurational audit tools (also called static audit tools) are those which make no changes whatsoever to the security state of the system. By examining the configuration of the target machine, passive tools attempt to infer the presence of configurational vulnerabilities [55, 3.1]. Furthermore, passive tools do not attempt to remedy those flaws that are detected.

[2] As an example of this, Unix's `/etc/hosts.equiv` file requires protection from unauthorised modification, as well as a check of its contents to ensure that no unauthorised hosts are trusted by the tested machine.

Conversely, active tools test for the presence of system vulnerabilities by attempting to exploit them from an attacker's viewpoint. For example, an active checking tool may test for access control weaknesses by attempting to log into the system using standard user accounts and guessed passwords. Tools that reconfigure the system in order to reduce vulnerability can also be classified as active tools, since such changes alter the security state of the system.

Active tools are potentially more dangerous than passive tools, as there is a chance they could be used by a real attacker to streamline the process of circumventing system controls. On the other hand, certain vulnerabilities may not be easily detectable without active testing.[3] Active tools may also be more system specific than passive tools, relying (for instance) on the presence of specific system version bugs.

ANALYSIS DEPTH

The cumulative effects of multiple vulnerabilities are sometimes an important configurational audit factor. Often, isolated system vulnerabilities do not present a large individual risk. However, when the effects of a number of these weaknesses are combined and then taken into consideration, the overall impact and hence level of risk may be much greater [55, 3.2]. Configurational audit packages can therefore be divided into those that check for *single* vulnerabilities, and those that check for *multiple* vulnerabilities.[4]

For example, a Novell NetWare file server allows users to log in as GUEST. This account is unprivileged, and given its limited access, does not represent much risk. However, on this same system, the SUPERVISOR account has an unprotected LOGIN script. An attacker using the GUEST account can insert commands in this script, which are executed when the SUPERVISOR next logs in to the server. These commands will grant SUPERVISOR privileges to the GUEST account. Hence, the combined effect of these two vulnerabilities is that an unknown attacker can gain total access to the file server.

Single vulnerability checking is straightforward to achieve, and usually consists of individual checks for particular vulnerabilities. Multiple vulnerability checks, on the contrary, tend to be more complex to implement. Tool implementors need to model the implications of combinations of individual vulnerabilities, and how these cumulative effects might lead to greater overall levels of system vulnerability. This is an ideal candidate for

[3] For example, checking for NetWare user accounts for NULL passwords is most easily accomplished by attempting to directly verify the password on file server on which that account resides.

[4] Polk refers to these as *single* and *system* vulnerabilities [55].

artificial intelligence techniques, such as expert systems. The COPS package includes an expert system multiple vulnerability checker, called Kuang.[5]

DISTRIBUTED TESTING

Configurational audits may take place over a network of connected computer systems. Tool implementors may choose to approach the issue of testing from a network oriented viewpoint, or from an isolated individual system viewpoint. Networked systems have different requirements in terms of vulnerability checking, which are outlined below:

- *More points of vulnerability*
  In a distributed system, there are more points of vulnerability to address. For example, as a file traverses a wide area network, it may be processed by four or five computer systems along the way. Each one contains potential vulnerabilities that may affect whether that file reaches its destination unaltered (or at all).

- *Trust relationships may be operative*
  Configurational audit packages need to check that trust relationships between networked systems are acceptable. Trust relationships are common in Unix networks, and allow systems to share information and processing with a conveniently low level of authentication.

  The implications of trust are important for configurational audit tools. An analysis may be carried out of a system that provides access to another system via a trust relationship. Even though the analysis may reveal no vulnerabilities within the tested system, vulnerabilities may still occur because of security flaws in the trusted partner. In order to prevent these vulnerabilities, the tool needs to recognise such relationships, and recommend testing of the trusted partner on a concurrent basis.

- *More chance of external attacks.*
  The connection of organisational computer systems may increase the population of potential external (non-organisational) attackers. Machines that are accessible from public networks may require especially stringent testing of network access controls and services.

One possible advantage of networked configurational auditing is that computationally intensive processing can sometimes be distributed among networked systems. For instance,

[5] Kuang is discussed in more detail in the next chapter.

the `crack` password cracker can be run concurrently on a number of hosts, effectively reducing the amount of time it takes to crack password files (see Muffett [42]).

REPORTING POLICIES

Configurational audit tools need to address the type and level of reporting that the system will support. Three issues are important:

- *Level of explanation*

  Existing configurational audit tools differ in their approach to how much explanation the package should provide about vulnerabilities that have been located within a system. The package may provide a complete explanation of the vulnerability, including how to remedy it, or it may simply mention that a vulnerability exists. As discussed in *Tool Misuse* on page 52, terse strategies tend to favour potential attackers, rather than those responsible for system security.

- *Vulnerability magnitude*

  Providing information about the seriousness of a discovered vulnerability is a very useful feature. Such guidelines help administrators or auditors filter tool output, and assist prioritising flaw safeguarding activity.

- *Fix or warn*

  Some configurational audit tools may provide an automated fix feature. For example, the ATP change detection tool (mentioned in [34, p.9]), incorporates a feature called *action lists*. On detecting a problem file, these can automatically change its ownership to root, thus preventing normal system users from accessing the file.

  As noted in [34], such active systems are not without risk. For example, the system may make mistakes that could disable the system, or could accidentally "fix" something that does not require fixing.

LANGUAGE CONSIDERATIONS

Most configurational vulnerability checks consist of relatively simple tests, which are often able to be implemented with system command scripts (such as Unix shell scripts or DOS batch files). In order to improve performance, or to gain access to information not available to shell scripts or batch files, some configurational audit tools may be partially implemented in a higher level language, such as C, C++, Rexx, Prolog, or Perl. At the most extreme end of the scale, some configurational audit tools are implemented entirely in high level languages.

Command scripts tend to be more accessible to the average system administrator, and lend themselves to easy modification, expansion, and portability. Conversely, tools written in languages such as C are harder to understand and modify, but offer vastly improved performance, and are able to accommodate more complex computations.

Selecting a suitable language for configurational audit tool components depends largely on four issues:

- *Target environment.*
  Some environments, such as Unix, store all security configuration information within human-readable text files. Provided these files are accessible, a large amount of tool analysis may be performed by scripts using relatively simple text processing utilities, such as Unix's awk, grep and sort. Some security information may be additionally available from system commands. In other environments, such as Novell NetWare, some security relevant information is only available via calls to system API's. Hence, such information is only accessible from within applications (such as SYSCON), or courtesy of specially written "helper" applets.[6]
- *Tool performance or functionality considerations*
  Some checks performed by configurational audit tools, such as generating file signatures or cracking passwords, are computationally intensive. Both tasks require levels of performance (or complexity) not well handled by shell script languages. The type of check performed by the tool may also influence the selection of language. For instance, expert system components may be implemented in a language such as Prolog.
- *Presentation requirements*
  There may be a requirement for presenting security information or analysis reports in a graphical format. Graphical user interfaces are generally inaccessible to command line scripting languages. One possible approach is to write a graphical front end that drives the underlying scripts and programs that perform the data gathering and vulnerability tests.
- *Distribution requirements.*
  Where a tool contains confidential or proprietary code, the only option available to the developer may be a compiled language, such as C.

[6] Helper applets are small applications that gather system information via standard API's, and then output this to a text file in a format able to be processed by other applications.

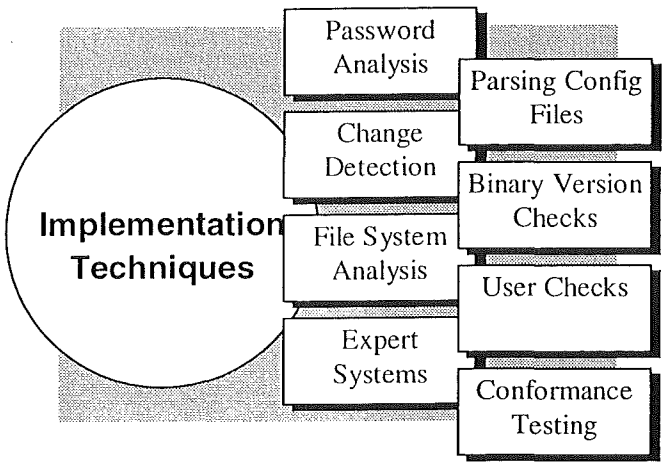


Figure 5 - 1 Configurational analysis implementation technique overview.

In practice most tools will use a combination of approaches, where whatever programming language is best suited for a particular task is selected. For example, COPS [23] contains components written in C, shell scripts, and PERL.

5.3 IMPLEMENTATION TECHNIQUES

The various implementation techniques used by configurational audit tools are summarised in Figure 5 - 1 above. These are discussed in more detail in this section.

ASSESSING CONFIGURATION FILE OBJECTS

Configurational files are used by systems to control core system services, user environments, network services, applications, and system startup behaviour. Weaknesses contained within these files, or of the safeguards protecting those files, may lead to vulnerabilities that are able to be exploited by an attacker. Both COPS and TAMU support configuration file checks.

There are two varieties of configuration file object: configuration files, and configuration scripts. Configuration *files* contain setup information for a system service or application, while configuration *scripts* contain system commands, and are run when system services are initialised (such as at boot time). Configurational files and scripts are attractive targets for attackers, because scripts are often run at elevated (system or root) privileges, while files may contain settings that enforce security for system services.

A configurational audit tool is concerned with three main configuration file issues:

- *Unprotected files*

  Some configuration files may contain information that is sensitive, or which if compromised, could by used by attackers. For example, the AUTOEXEC.NCF file

used during booting of a NetWare file server may specify a password on the command line used to invoke the remote console utility. By obtaining this password, attackers may be able to remotely compromise the security of the server console. Configuration files of this nature should be protected against inadvertent disclosure.[7]

Configuration files that are subject to unauthorised modifications present an even greater risk. For example, a Unix attacker who is able add entries to another user's `.rhosts` file will quickly be able to compromise that user's account.

- *Unprotected referenced files*
  Configuration files that depend on or invoke other files that may be subverted by attackers present the second source of configuration file vulnerability. Figure 5 - 2 shows a scenario in which a privileged configuration script calls three sub-commands: *B*, *C* and *D*. *B* in turn calls *E* and *F*. As can be seen from the diagram, configuration file *E* is not appropriately protected, and (in this example) has been modified by an attacker to insert a trap-door in the login binary. When the top level configuration file is executed, the attacker's additions to E will be executed at whatever level of privilege the top level script was invoked with.

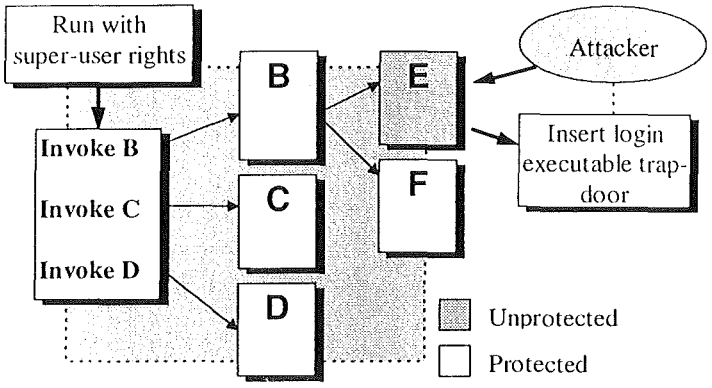


Figure 5 - 2 Sample configuration file interdependencies.

Configuration file interdependencies such as these are very difficult to detect without some form of automated tool that recurses such file dependency structures.

- *Configuration file contents*
  While the two concerns above deal with the protection of file objects that contain configuration information, settings specified within file itself may also be a source of vulnerability.

  Under Unix, for example, most system services are configured via commands or settings contained in standard text files. Incorrect settings in those files may cause

[7] An even better approach is to avoid placing passwords in configuration scripts or files. However, this may impede automated boot sequences.

configurational errors. For many years SunOS Unix machines were installed with a default + + entry in the `equiv.hosts` configuration file. This entry allows any user on any machine to bypass authentication mechanisms during remote logins.

A configurational audit tool can check to ensure that such defaults and weak settings are not present in the configuration of the machine. Settings files checked could include system services, network services, application resource files and user login files.

Detecting unprotected file and referenced file vulnerabilities is a relatively straightforward process. Usually, such checks will involve a simple test of the access permissions of top-level configuration scripts, and a recursive check of the files that these scripts reference (if any). This is a general process that can be applied where necessary to any configuration file.

Assessing the implications of *settings* within configuration files is a more challenging problem. There are a vast range of configuration files on a standard Unix machine, and each one may control a different system component. In addition, different platforms will have various configuration file peculiarities.

In order to assess the security implied by these files, the configurational audit tool requires specific information about which settings in which files can affect system vulnerability.

PASSWORD ANALYSIS

Weak password choices on the part of system users are a common problem at many sites. Users may select passwords that are some permutation of their login name, their family name, the name of their cat or spouse, or a word that is available in any one of hundreds of on-line dictionaries. Such passwords may be either easily guessed by an attacker, or may be guessed with the help of a password cracking program.

Factors such as: password length, frequency of change, the locatability of passwords in common dictionaries, and where passwords are stored, can all have an impact on the configurational security of the system. These factors are good candidate checks for a configurational audit tool. [55, 2.2.1].

Several password cracking programs exist, the most commonly available being `Crack`, by Alec Muffet [42]. `Crack` works by using words from standard dictionaries, as well as lists of commonly used non-dictionary passwords, to generate password guesses for each user account. The password guess is encrypted using the same method used to encrypt system

passwords, and the encrypted value is compared to the value stored on the system. If a match is found, then the guess was correct, and the password for that account compromised. `Crack` is perhaps one of the most sophisticated password checkers available, as it can generate an almost unlimited number of permutations of each password guess. Additionally, `crack` is able to distribute processing around a number of hosts on a network.

Password cracking is a useful addition to configurational audit, especially considering most attackers will already possess such a tool.[8]

FILE SYSTEM ANALYSIS

Discrepancies and errors in the configuration of file system access controls may lead to inadvertent modification or disclosure of information stored on the system. File system analysis may consist of two types of check:

- *Corrupt or suspicious file checks*
  This check is performed by searching system and user areas of the file system in search of files and directories that may indicate that there are file integrity problems, or that an attacker has been active on the system.

  For example, the tool may check to ensure that configuration, binary and log files have not been corrupted. Alternatively, the tool may search for files and directories that have been intentionally hidden (such as a "..." directory in Unix), or files that represent security hazards (such as a SUID root copy of the csh command shell).

- *Access control check*
  Access control checks examine vital system and user directories to ensure that no unauthorised users are able to make changes to files within those directories. For example, checks may ensure that system binary files are only updateable by system staff. Tests may be expanded to include access control checks for application binaries, and a general check of user access and rights within the file system.

Both COPS and TAMU include these checks as part of the system.

[8] Other approaches are not precluded by the use of password crackers. For instance, password changing programs that reject weak password selections are one option, as are shadowed password files, a feature available on some versions of Unix which hides password information away from unauthorised users.

CHANGE DETECTION

A useful configurational audit technique is *change detection.* The reasoning behind adopting a change detection approach is relatively simple - many *introduced* vulnerabilities in systems manifest themselves as unexpected changes in system objects. Comparing a previously stored "snapshot" of a secure system to the current configuration, and checking the results of this comparison for unexpected changes, may indicate the presence of an intruder, a virus, or some other inserted vulnerability. Change detection systems can also detect when objects are added or removed from the system.

An added advantage of change detection techniques is that after a system has been attacked, system administrators can determine which (if any) files were affected by the attack. In these cases, only those files need to be restored from backup media. In contrast, without change detection tools, the system administrator can no longer trust any system files, and is forced to restore the entire system from backup media.

In order to be effective, change detection systems should be used and maintained on a regular basis. The Tripwire [34] package is based on change detection techniques, while COPS supports rudimentary change detection.

Selecting signature algorithms

A major step in implementing a change detection system is selecting a suitable method whereby changes may be detected. As noted by [34, 2.4.1] two approaches are feasible:

- *Compare the current object with a "known" saved copy.*
  With this approach, a complete copy of the object to be compared is stored (preferably on removable or read-only media). During checking, a byte by byte comparison can be performed between the current version and the saved version.

  The main advantage of this technique is that if changes are detected, the auditor can determine exactly what the changes were by comparing the original and changed versions. The main drawback is the space that is required to store the saved information, which could potentially double storage requirements if entire file systems are tracked.

  Nevertheless, this may be an appropriate method if the amount of information that is to be tracked is small. For example, tracking user account configuration changes is possible using this method.

- *Generate an object signature.*

  For large system objects, such as binary files, a much more sensible approach is to record a "signature" of the original file, which is compared to the current signature. The advantage of this approach is that space requirements are minimised. The disadvantages are that if the signature is not secure enough, this approach may be spoofed. Additionally, this method can only detect that a change has occurred, and not the nature of the change. However, where space is at a premium, and large system objects are being tracked, this is usually the best approach

The simplest form of signatures available are checksum algorithms. As an example, an algorithm could add the numeric value of bytes contained in the file, and then take some function (for instance modulo 256) to produce a single byte signature.

Unfortunately, this algorithm is inherently insecure. All an attacker need do is calculate the signature before the change, alter the file, then calculate the byte value required to generate the same signature. This (brute-force) attack is surprisingly effective, even against more sophisticated signature algorithms such as CRC's. For a signature n bits long, an average of 2^{n-1} attempts will be required to find an appropriate offset value that would generate the required signature [34, 2.4.2].

Two different objects producing the same signature is known as a collision. Generally, the longer the key, and the more effective the signature algorithm, the fewer undesirable collisions will occur. Brute force attacks are more effective where the signature algorithms produce many collisions (as in the case above).

Another approach may also be used to spoof the signature algorithm. By reverse-engineering the signature algorithm, an attacker could create a reversed version of the function that could be used to generate an arbitrary file that produces the required signature [34, 2.4.2]. As mentioned in [34], CRC functions (often used in file integrity products) are not immune to this type of reverse engineering. Obviously, in order for change detection systems to provide an acceptable level of assurance, a stronger signature algorithm is required.

Message digest functions are one such approach. These one-way hash functions are algorithmically difficult to invert, and produce few collisions. These two features make such

functions good choices for authentication signatures. Examples of such functions include MD-4, MD-5 and Snefru.[9]

Storage and updating strategies

When designing a change detection system, consideration should be given to what information should be saved, how signatures are to be stored, and how such signature databases are to be updated. Signature databases need to be generated when the system is initially installed, and thereafter whenever authorised changes are made to tracked system objects.

Signature databases may be stored as binary only, or in human readable form. The advantage of storing signature databases in human readable text files is that it is possible for the administrator to replace or remove individual entries. TripWire, for example, supports the use of an independent program to generate a signature, which is then manually placed in the signature database file. Binary databases, on the other hand, may offer better performance.

Updating signature databases

Given a large number of tracked files, generating signatures with computationally intensive algorithms such as message digests may take considerable amounts of time. An ideal solution is to enable signatures for individual files or sets of files to be generated, rather than having to regenerate them for the entire database every time a change is made to a subset of tracked system objects.

As examples of update strategies, TripWire supports updates of specific files, or entire directory subtrees. This may occur on an ad-hoc basis, or as an update during a check. On the other hand, the `crc_check` program included with COPS requires that the entire database be regenerated when changes occur.

With respect to the file system, the issue of whether to save directory structures or not is also important. In some cases, detecting the addition or removal of directories is just as important as detecting where files have been altered.

[9] References and collision comparisons for these algorithms can be found in [34].

Signature database storage

Storing the signature database in a secure location is also very important. Attackers who can modify the database can cover their tracks by surreptitiously updating the signature database after making a change.

Storing signature databases on removable media, such as tape or floppy is a possible solution. Unfortunately, this may limit automation of the package, in that those media have to be available during checks. Ideally, a copy of the database should be stored on a read-only media (such as WORM drives or read-only network file systems), so that it may be accessed, but not changed [34, p.4]. However, updates of signature databases stored on read-only media may involve temporarily storing the database in a secure writable location.

Change detection drawbacks

Although change detection is a useful technique to include in configurational audit tools, several possible drawbacks of using such systems exist, which preclude their exclusive use as a tool for detecting configurational vulnerabilities. Bontchev [5] identifies two possible problems:

- *Initial system state*
  In order for change detection systems to effectively detect unauthorised changes, they must be installed on a secure system. If the system is not initially secure, then signatures of already compromised system objects will be stored as part of the supposedly clean change detection profile.
- *False-positive alerts*
  False positive alerts may occur when the change detection system detects an authorised change to a tracked system object. For example, legitimate reasons for changes are software upgrades, reconfiguration of existing services, or natural changes in access rights of tracked users. These create noise which must be filtered out later in the reporting process.

BINARY VERSION CHECK

System binary files may contain security bugs and shortcomings that have been addressed with vendor supplied updates. Useful additions to configurational audit tools are checks that confirm whether such updates have been installed, or alternatively, whether the version installed on the system contains known vulnerabilities.[10] Particular attention should be paid

[10] A slight variation on this theme are virus checkers, which examine executable code for recognisable virus signatures.

to "high risk" binaries, such as login authenticators, encryption utilities, and network service programs.

By their nature, version checks are extremely platform specific. The TAMU system solves this problem by supporting a set of comprehensive system specific version checks for a variety of platforms. Three approaches to checking system binaries are available

- *Comparing file date stamp information*
  This approach compares the creation date and time of the binary file to that of either the patched version, or of versions that contain known vulnerabilities. When an obsolete or vulnerable version is found, the program alerts the auditor.
- *Comparing file signatures*
  The second approach involves comparing the targeted binary file's "signature" to that of updated or vulnerable binaries. The signature is an algorithmically generated function of the contents of the binary that, depending on the algorithm, is unique for most files. Signature algorithms employed include CRC functions, and one-way hash functions such as MD-4, MD-5, or Snefru.[11] This is a more secure approach to that of comparing file dates, due to the ease with which these are changed by attackers.
- *Exploiting the vulnerability*
  An active checking system may verify the existence of a known system binary flaw by attempting to exploit it. This will reveal whether or not the erroneous binary is installed, but unfortunately may also lead to unwanted side-effects, or may change the state of the system.

USER CHECKS

The security of individual user accounts may be of interest to configurational audit tools. Given the problems associated with poor user security habits, checking user settings may be an important facet of such a tool. Checks could include password strength tests, login and resource file checks, as well as other checks examining individual user access rights to file system objects, hardware devices, and network services.

Especially important on Unix systems is checking that user PATH environment variables specify system directories before personal, public or the current directories. This helps reduce the risk of Trojan and "companion" type programmed attacks.

[11] These signatures were discussed in more detail on page 69.

Depending on the target system, these checks may be handled entirely by parsing the contents of user or system configuration files (as is the case for Unix). Alternatively, they may consist of checks performed by applications interrogating the system for security relevant information (as is the case with NetWare).

CONFORMANCE TESTS

While the primary consideration of a configurational audit tool may be to detect configurational flaws, another goal (as discussed on page 48) may be to enforce security mechanism usage standards according to organisational polices.

This may be achieved by comparing system objects to some ideal "baseline" or "template" object. For example, a template may specify that users in a particular group should not be granted access to certain portions of the file system.

One advantage of conformance testing is that it can be used to customise minimum security standards on a site, or organisational, basis. These customisations can be based on perceived threats, site environments, or the organisational security policy.

However, this configurability may also present problems. For instance, if insecurely specified, a baseline may not even offer an acceptable minimum level of security. Hence, attention to detail is required when assessing baseline requirements.

EXPERT SYSTEMS

Configurational audit tools may make use of artificial intelligence techniques in order to detect configurational vulnerabilities. Techniques that may be applicable include:

- *Expert systems*

  Expert systems may prove to be particularly valuable. For example, the COPS package includes the Kuang expert system, which is used to assess account security. Given a user name, Kuang is able to assess whether that user account is able to be compromised.

- *Neural networks*

  Neural networks may be trained to recognise certain configurational patterns that lead to vulnerability. The biggest barrier to using neural networks will be that of effectively training them to recognise such configurational errors.

5.4 CHAPTER SUMMARY

This chapter has examined configurational audit tool design philosophies, as well as several specific implementation techniques that are used in such tools. As discussed, a number of philosophical issues must be considered before implementation of a tool can begin; these issues help shape the final form of the tool, and how it will be applied to systems.

The majority of tests performed by configurational audit tools consist of relatively simple checks. The next chapter reviews a selection of existing tools that implement some of these techniques.

CHAPTER 6
A SURVEY OF TOOLS

This chapter surveys some commonly available configurational audit packages. Included in the review are COPS, ISS, TAMU, and Tripwire.

6.1 COPS

The COPS (**C**omputerised **O**racle and **P**assword) **S**ystem, developed by Farmer *et al* [22, 23] in 1989 represents one of the most mature Unix vulnerability checking systems currently available. Freely distributed around the Internet since 1989, the portability and availability of COPS has ensured its proliferation throughout the Unix community.[1]

A general-purpose passive security checker, COPS features checking of the target host's system configuration, user account settings, compares system executable file dates to those mentioned in CERT advisories, and provides rudimentary file integrity checking.

THE COPS PHILOSOPHY

COPS was developed with two overall objectives in mind (Farmer [22]): First, to help foster and develop an understanding within the Unix community of the problems associated with Unix security, and second, to develop a tool that could be used to inform Unix system administrators of vulnerabilities present in their system. Some specific design goals of COPS, described in [23], are outlined below.

- *Configurability*

  COPS is designed to be easily configured and extended to suit local conditions and security needs. Accordingly, much of the functionality of COPS is contained in Unix shell scripts. Where necessary, C programs are provided to speed up computationally intensive functions, such as password cracking, or to perform complex tasks such as determining writability of directories and files.[2]

[1] Version 1.04 of COPS was reviewed.

[2] The current distribution of COPS includes PERL versions of the scripts (p-COPS), which are faster, and allow for more complex checking than a standard shell script.

- *No changes*

  As a passive tool, COPS makes no changes to the security state of the target system, nor does it attempt to repair security vulnerabilities that it finds. This allows most COPS components to run at non-privileged levels, reducing the attendant risks of subverted versions of COPS being used to attack systems.

- *No Explanations*

  COPS provides no explanations of vulnerabilities found in the target system. This is a conscious decision by the COPS designers to reduce the chances of alerting crackers to the existence of holes of which they may otherwise be unaware. COPS simply notes that the vulnerability is present, and issues a warning.

- *No cracking assistance*

  In order to reduce the risks of COPS being used to attack systems, the designers of the package ensured that it would contain no components that be of significant use to an attacker. Where possible, COPS uses tools and libraries that pre-exist on the target system.

- *Ease of use and comprehension*

  A handful of easily understood Unix shell scripts provide much of the functionality of COPS. Modifications of these scripts supporting localisation requirements, or expanding the range of checks performed by COPS, are relatively straightforward. As an added advantage, scripts can be easily scanned for subversive code, and are particularly portable.

  Using the shell scripts is also straightforward, as in most cases the default script options should prove to be sufficient. The scripts are well documented, and can be called from a single 'super-script'. Each script can also be used for stand-alone checks.

COPS IMPLEMENTATION

Analytical components of COPS work by parsing system and user configuration files, and noting where the contents of those files may lead to security problems. As with many Unix programs, COPS is made up of a number of smaller, specialised sub-components that work in concert to check the security status of the system. The majority of COPS is implemented as shell scripts, although several C programs are included where performance is required. COPS components are summarised in Table 6 - 1 on page 77.

Script	Actions Performed
bugs.chk *(script)*	Performs architecture (or vendor) specific script to determine if any known security bugs are present in system executables. COPS uses information from CERT advisories to select executables to check. There is no established update strategy to maintain up-to-date checks.
cron.chk *(script)*	Checks cron configuration files for potential danger. Permissions of the cron files are checked, and the contents of those files are parsed for world writable paths.
dev.chk *(script and C)*	Check devices mentioned in `/etc/fstab` for world writability, and the directory exports mentioned in `/etc/exports` for appropriate restrictions. Of particular interest are file systems that are exported without restrictions regarding which hosts may mount them.
ftp.chk *(script)*	Assess ftp configuration, with a particular emphasis on the security of the anonymous ftp service. Check protection of configuration files, and check for several common anonymous ftp configuration errors (such as writable ftp directories).
is_able.chk *(script and C)*	Uses the `is_able.lst` configuration file to assess the world or group writability status of important system files, directories, and other (user defined) areas of the file system.
pass.chk *(script and C)*	Attempts to guess user account passwords. A list of common passwords used for guesses are included in the COPS distribution, and standard dictionaries can also be used. This program uses an accelerated version of the Unix `crypt()` function to speed up the password cracking process.
rc.chk *(script)*	This script parses `rc.*` files in the `/etc` directory for paths and filenames that are world writable. Commands in these files are normally executed when the machine boots.
root.chk *(script)*	Checks the root account for vulnerability. Checks root startup files for writability, writable directories on root's path, non-root `.rhost` entries and assorted other items.
suid.chk *(script)*	Searches the file system looking for changes in files with the SUID or SGID bits set. Checks for unusual directory or file names. A list of known SUID programs is kept in the COPS directory, and is used to determine when changes occur, which are detected by comparing directory listing information. COPS makes no attempt to algorithmically determine if the contents of the file have changed.
user.chk and home.chk *(C)*	Checks that user home directories are not world writable, and that key files contained in home directories are not world writable. Files checked include `.profile`, `.login`, `.forward`, `.rhosts`, `.emacsrc` and `.dbxinit`.

Table 6 - 1 COPS scripts and programs.

```
**** ftp.chk ****
Warning! root should be in /etc/ftpusers!
ftp-Warning! /home/ftp/etc/group and /etc/group are the same!
ftp-Warning! Incorrect permissions on "ls" in /home/ftp/bin!
ftp-Warning! Incorrect permissions on "passwd" in /home/ftp/etc!
ftp-Warning! Incorrect permissions on "group" in /home/ftp/etc!
```

Figure 6 - 1 Sample output from the ftp.chk COPS script.

Reports produced by the COPS system are stored in a text file for perusal by the administrator or auditor once the run is complete. Sample output, from the *ftp.chk* program is shown in Figure 6 - 1.

The SU-Kaung expert system

Included in the distribution of COPS is the SU-Kuang expert system. This system, described by Baldwin in [3], allows the COPS user to assess whether an account may be compromised, given access to a set of accounts or user groups on the target machine.

SU-Kuang uses a set of rules that model Unix protection mechanisms. Starting with a goal (i.e. "compromise the root account"), SU-Kuang checks its knowledgebase for a rule (or a set of recursively called sub-goals) which would be sufficient for it to achieve that goal (ie re-write `/etc/passwd`).

By checking the protection configuration of the target machine, and by using an initial level of user access, SU-Kaung can decide if it may apply a rule in order to achieve its goal. If the overall goal proves to be attainable, SU-Kaung can provide an explanation of the chain of events that could occur to compromise the target account.

CHANGE DETECTION FEATURES

COPS includes a component for checking system executables and configuration files for changes. The *crc.chk* script generates a database of file information, which includes directory list information and a CRC value based on the contents of each file. Files to include in the CRC database are specified in a configuration file in the COPS directory.

Periodically, the user generates a new database, which is compared to the old CRC database by the *crc_check* program. Changes in the contents, date, or permissions of files are flagged for perusal by the system administrator.

COPS COMMENT

Overall, COPS is a comprehensive package that covers a lot of areas. COPS benefits from extensive use and peer review within academic and commercial organisations around the Internet. Some aspects of COPS warrant further discussion:

Vulnerability Explanations

As discussed on page 76, COPS does not provide explanations of vulnerabilities that it finds. In contrast, the TAMU system (described in Section 6.3 on page 84), is able to notarise vulnerabilities detected with an explanation of the nature of the problem.

The usefulness of such a "security through obscurity" policy as embodied by the COPS approach is questionable. In practice, it is often the cracker who is better equipped with system specific knowledge about vulnerabilities than the legitimate system administrator. The "no explanation" policy may simply serve to keep these administrators in the dark about the seriousness (or otherwise) of the vulnerabilities found by COPS.

It is curious to note that the COPS distribution includes reasonably detailed explanations of COPS errors and warnings in a separate file.

Problem Magnitude

In COPS, there is no concept of problem "magnitude". When a configurational error is detected, it is simply flagged as a warning (see Figure 6 - 1). There is no indication of the seriousness or otherwise of a flagged problem. This could make prioritising administrator responses to COPS messages a difficult task.

In addition, where there are a large number of "noise" warnings, it is possible to miss serious problems. An additional field for each message reflecting the seriousness of the problem would be a useful addition to COPS reports.

Integrity Checking

COPS integrity checking (see the crc_check discussion above) suffers from two main problems. First, the choice of a weak check algorithm (a relatively simple CRC function) means that a determined attacker can remain undiscovered by ensuring that changes made to system files and binaries produce the same signature. As Kim and Spafford note in [34]:

> *"... Reversing the CRC function to yield a desired signature is a well-understood process, and tools to assist a potential intruder are widely available..."*

The second problem with the COPS approach to integrity checking (also mentioned in [34]) is that it is impossible to regenerate a single database entry without regenerating the entire database. For large databases, this could represent a significant amount of time.

Bug Checking and file tracking

Some of the checks performed by COPS use untrustworthy means of gathering data. For example, *bugs.chk* uses file timestamps to decide whether patched system binaries have been installed. Timestamps are again used to assess whether SUID and SGID files have been adjusted. This is an inherently insecure method of checking for patch installations or file changes, as timestamps are easily modified by an attacker.

A better approach is to calculate a file signature from the contents of the file being checked, and then compare this (in the case of system binaries) to the known secure version. In the case of tracked SUID or SGID files, the comparison should be to a previously saved file signature. Algorithmic signatures were discussed in *Change Detection* in the previous chapter

6.2 ISS (INTERNET SECURITY SCANNER)

ISS is one of the few active vulnerability testing systems presently available. Developed by Klaus [37] circa 1991, ISS identifies whether a host has any one of several well-known network oriented vulnerabilities. ISS has been the subject of at least one CIAC bulletin since its release onto the Internet.[3, 4]

ISS PHILOSOPHY

An active checker, ISS adopts a doorknob-twisting approach to vulnerability testing by simulating an attack by an external user. In the course of an analysis, ISS tests candidate network services in the same way a human attacker might. Thus, ISS attempts to telnet to the target, tests the setup of ftp, attempts to communicate with network services, and gathers information about the target host using standard Unix network utilities. With the appropriate tools, ISS can even attempt to acquire the `/etc/passwd` file from the target host.

[3] See [15] for CIAC security advisory on ISS.

[4] Version 1.21 of ISS was reviewed. ISS is available from several Internet sources, most notably the comp.sources.unix archives.

ISS is primarily a tool for assessing the *network* vulnerability of the target system, and all attacks take place using network services. Other issues, such as file system security, user account integrity and so on, are not considered by this package. Four other ISS philosophical issues are worthy of note:

- *Gathering Information*
  ISS collects considerable amounts of information from the target system. In particular, lists of active users, exported file system information, network service availability and information about supported network services are collected. Much of the data gathered by ISS is useful to an attacker looking for obvious defects in the configuration of the target system.
- *An External Perspective.*
  ISS checks vulnerabilities from the perspective of an external attacker. The ISS tool does not require that the attacker already have an account on the target system. ISS assumes an external attack methodology in each of the hosts it targets.
- *Generic Network Tests*
  Some vulnerability analysis applications test for vulnerabilities that are specific to a particular version of Unix. ISS looks for generic Unix configuration errors that may lead to, or indicate, vulnerabilities.
- *Scanning For Targets*
  ISS is able to scan a range of IP network addresses, specified by the user, in search of target hosts. Thus ISS may be used to check a network domain which may be unfamiliar to the ISS user. Additionally, this feature allows ISS to automatically test a large number of hosts. Both factors may encourage the use of ISS as a "blind" attack tool.

ISS IMPLEMENTATION

ISS is made up of a single C program, which performs all checks. Other standard Unix utilities, such as `rusers`, `rcpinfo`, `showmount` and `ypwhich` are called during ISS's analysis.

When instructed, ISS will scan a network domain sequentially for hosts. For each host found, ISS performs a series of checks. ISS uses command line options to specify which checks to perform, and to specify where to store ISS output. The checks and information gathering that ISS performs are outlined in Table 6 - 2 on page 82. Figure 6 - 2 shows sample output from the ISS package.

Check	Actions Performed
Telnet check	1. Attempt a basic telnet connection to the target host. Log welcome screen to file. 2. If a login prompt is acquired, attempt to log in as `sync`. 3. If a `sync` login is successful, screen information from the login process will help reveal additional facts about the target system, such as the release version of the operating system, and perhaps a message of the day. Capture information to log file.
Sendmail and Mail aliases Check	1. Attempt a connection to the SMTP port. If successful, log received information. This may include the host name, the type of operating system and (occasionally) the version of `sendmail` that is installed (Sample output from ISS's sendmail interrogation is shown in Figure 6 - 2). 2. Verify the existence of system accounts `guest`, `bbs` and `lp` using the SMTP `VRFY` command. (these should generally be disabled accounts). Log the results. 3. Check if the mail aliases `decode` and `uudecode` (which may cause vulnerabilities if improperly configured) are enabled. Log the results. 4. Try the `wiz` and `debug` sendmail commands. Log any results.
FTP check	1. Check if FTP is enabled on the host, and that anonymous logins are allowed. Log the results. 2. If anonymous ftp is running, attempt to create (and remove) a test directory. If ISS can perform this step, then the ftp directory is world-writable, and the host is vulnerable to attack. Log the results.
RPC Check	1. Use the `rpcinfo` utility to gather information about the remote network services that the host provides. Log received information, and then parse it for common services such as ypserv, mount, rexd, rusers, etc. 2. If `ypserv` is running, attempt to construct the domain name of the target host, based on the machine name, or from information from the `smtp` check. 3. If the domain name address is correct, optionally try to take advantage of a `ypserv` bug to grab the password file from the target system. (This requires an additional utility not shipped with ISS). 4. For other services, collect and log useful information. For example, if the mount service is running, use the `showmount` utility to show the file systems that the target system exports. If the `rusers` server is running, get a list of the currently active users.

Table 6 - 2 ISS Active checking strategies

ISS COMMENT

From the vulnerability analysis point of view, ISS is an interesting system simply because of its active approach to testing host vulnerability. Whether or not this active approach is better than passive vulnerability analysis methodologies is an open question. Certainly, it would be difficult to evaluate active testing methodologies based purely on data collected from the success or otherwise of ISS, because of the relatively simple nature of ISS attacks.

Because of its active nature, ISS leaves traces of its activities that are detectable by alert system administrators. For instance, during testing of ISS, SMTP port attacks (see Figure

```
SMTP:220 cosc.canterbury.ac.nz Sendmail 4.1/SMI-4.0 ready at Mon, 2 May
94 18:00
550 tonysg... User unknown     (<--returned from VRFY guest)
550 decode... User unknown     (<--try decode     )
550 bbs... User unknown        (<--try bbs        )
550 lp... User unknown         (<--try lp         )
550 uudecode... User unknown   (<--try uudecode   )
500 Command unrecognized       (<--wiz Command    )
500 Command unrecognized       (<--debug Command  )
221 cosc.canterbury.ac.nz closing connection
```

Figure 6 - 2 ISS SMTP port check output (author's comments in brackets).

6 - 2) were noticed by the system administrators here at the Computer Science department of Canterbury University. In this case, the departmental mailer-daemon automatically mailed a transcript of the failed session to the postmaster.

Perhaps the most common Unix system administrator objection to ISS is that it is a dangerous tool in the hands of an attacker. A typical scenario would be an attacker using ISS to remotely locate and gather information about vulnerable Internet hosts in a domain address range. Once attack targets have been selected using ISS, more sophisticated methods could be brought to bear to further compromise those hosts.

Another area of concern to administrators is that ISS attacks can be made from an external (and perhaps anonymous or remote) source. In contrast, COPS and the other packages discussed in this chapter need a valid account on the target system in order to function. Where other packages use methodologies revolving around prior knowledge and access to the configuration of the target system, ISS does not.

What is less obvious is that the distribution of the ISS source code represents a greater danger than simply the ISS program itself. The code is mostly straight-forward, easily understood and easily modifiable. Although any reasonably competent programmer familiar with Unix networking could reproduce the functionality of ISS, this code still represents a ready-made skeleton for more advanced (and perhaps hostile) mutations.

As a simple example, modifications could be made to ISS so that more default user accounts are tried during telnet port checking. More intelligence could be added to ISS to enable it to target specific versions of Unix, and to exploit known vulnerabilities in the

targeted version. Considering the ease with which the source code can be modified to include extra attack methodologies, it is perhaps surprising that more variants of ISS do not exist.

Note that currently the only ways to protect against ISS attacks are to either limit the network services that hosts export to the outside world, or to use firewalls to regulate external or untrusted network traffic. Additionally, packages such as `tcp_wrappers` can be useful in detecting ISS type attacks.

Chris Klaus, the author of ISS, is reportedly working on a commercial version of ISS that hard-codes the network address range of the customer's machines, and consequently cannot be used for ad-hoc attacks on machines. This version was not available for review.

6.3 THE TAMU SECURITY PACKAGE.

The TAMU Security Package (Safford *et al.*[65]) was developed at Texas A&M University after several machines at the university were compromised by coordinated attacks from Internet hackers. In response to these and other break-in attempts, Texas A&M University implemented a three-pronged strategy designed to improve overall campus network security. The TAMU "tiger" scripts, a static auditing system, comprise part of this strategy.[5]

THE TAMU PHILOSOPHY.

The security of the Texas A&M University campus network came under review after several attacks from the Internet in August 1992. During these attacks, it was discovered that hackers external to the university had compromised an unknown number of TAMU's hosts, at that in one case had even set up a clandestine bulletin board on a local host.

Faced with the prospect of investigating the effects of these attacks on a large number of Unix hosts, and a limited number of knowledgable staff, work was begun on a collection of utilities and scripts that could be used to automatically search for and report security problems on TAMU campus hosts. These become known collectively as the *tiger-scripts*.

An overall approach to security was developed that was based around three fundamental tools. First, a filtering bridge (called *drawbridge*) was installed that filtered all incoming and outgoing Internet packets. This filter controls connections on a per-machine and per-

[5] Version 2.2.3 of the TAMU tiger scripts was reviewed.

Error Class	Description.
ALERT	A positive sign of intrusion was found.
FAIL	The problem that was found was extremely serious.
WARN	The problem that was found may be serious, but will require human inspection.
INFO	A possible problem was found, or a change in configuration is being suggested.
ERROR	A test was not able to be performed for some reason.

Table 6 - 3 Tiger script problem classifications.

port basis, with connection decisions made according to security policy and individual needs. Using this package, network administrators can control which machines are able to receive incoming Internet packets, and what ports on those machines are available to external users.

Second, a set of specialised packet monitors was developed that attempted to detect intrusive behaviour or attempts to circumvent the Internet packet filtering mechanism. These monitors additionally log connection and packet information.

The third part of the TAMU Security Package is based around the tiger scripts. After 1992, development work on the scripts continued, to the point where the scripts are now used to certify machines for connection to the campus network and to test suitability for Internet access. A campus host wishing to gain access to the TAMU network must demonstrate that it successfully passes tiger configuration tests before it is allowed to be connected to the network, and hosts must periodically be retested to retain IP certification.

The Tiger scripts.

In many respects, the tiger-scripts are an updated and more closely integrated version of the COPS system discussed in section 6.1. COPS and TAMU share many common features, and perform similar checks.

Like COPS, much of the tiger's vulnerability analysis is performed by script files. Where necessary, C programs are used to improve performance, or to perform functions that stretch the limits of script programming. Several aspects of the tiger scripts are of interest, and are discussed below.

.Script	Checks
check_alias	Checks for passwordless accounts, home directory ownership and access permissions, ownership and access of configuration files and disabled accounts with .rhosts, cron entries or .forward files that execute a program.
check_anonftp	If the target host supports anonymous ftp, this script checks the integrity of it. Ownership and permissions of critical files and directories are checked, as well as ftp accessible writable directories.
check_cron	Checks user 'cron' files, examining permissions and ownership of pathnames used by each cron entry. Executables with relative pathnames are also checked.
check_exports	Checks the configuration of NFS exported drives, and flags those file-systems that are exported with root access.
check_group	This script cross-references sources of information for group configuration for inconsistencies.
check_inetd	Examines the /etc/inetd.conf and /etc/services configuration files looking for mismatched Unix services, and services that have been added in addition to those set up in the standard distribution.
check_known	Checks the system for signs of an intrusion. Checks include searching for unexpected files, and a check of directories (such as lost+found) that are known to be used by intruders.
check_path	Checks the PATH variable in various shell startup scripts. In particular, the script checks for a "." path entry, and checks the access and ownership of executables in the path. This script usually only checks the root user, but can be configured to check all user accounts.
check_perms	Checks the ownership and permissions of system files. This script takes input from a system specific database that tells it which paths to check, and what the expected permissions are.
check_rhosts	Checks .rhosts files in home directories. These files are checked for standard Unix problems (such as + entries, or untrusted host entries). A PERL version is also able to verify that remote users specified in .rhosts files are the same person as the local user.
check_signatures	Validates system binaries. This check uses a database file of stored binary signatures, to which it compares those binaries installed on the system. Any mismatches or obsolete binaries are reported. Signatures are calculated by the TAMU team, and are distributed in either MD-5 or Snefru format. Periodic updates are available.
find_files	Searches file systems for "problem" files, including: SUID executables, device files, symbolic links to system files, world-writable directories, files with an undefined owner or group and files with unusual filenames.
check_embedded	Examines files and extracts file and directory paths which appear to be embedded in the script. These paths are examined for proper ownership and permissions. Embedded paths in these referenced files are in turn examined for proper ownership, and so on until the script reaches a user defined recursion level, or it runs out of files to check.

Table 6 - 4 Tiger script summary

- *Platform Specific Checks.*

  Initially, the tiger-scripts were targeted at SunOS 4.1.x machines, but later versions have included support for other versions of SunOS, AIX, HP-UX, IRIX, Linux and UNICOS. Where the target operating system is not recognised, tiger falls back to a "generic" check of the system.

- *Signature Checks.*
  The tiger-scripts take the Unix variant and version of the target host more seriously than COPS. One of the features of TAMU that separates it from COPS is that it can algorithmically compare the system binaries against signatures for that particular operating system, and can detect whether the binary has been changed, or whether a particular security patch has been applied. Signatures currently exist for the platforms listed above, and are updated periodically.

- *Reporting*
  Messages generated by the tiger scripts are classified according to the type of error detected during the analysis. This makes searching for serious problems in tiger-script output more convenient. These error classes, from [65], are detailed in Table 6 - 3.

- *Explanation Policy*
  Where requested, the tiger-scripts will notarise problems found with a full text description of the nature of the problem, and the risks that the problem may present to a system.

- *No Changes*
  The current tiger-scripts are completely passive, and make no changes to the configuration of the target system.

- *Ease of Use*
  The tiger-scripts can be invoked by a main script, or can be called individually. Minimal setup is required, and all site-specific and configuration information required by the package are stored in one or two configuration files.

TIGER SCRIPTS IMPLEMENTATION

The tiger-scripts are usually invoked by a main script (called *tiger*), although each script is a standalone module in its own right. The tiger scripts are summarised in Table 6 - 4 on page 86.

TAMU COMMENT

On the whole, the TAMU checker represents a more advanced, integrated, and streamlined checker than COPS. COPS is the result of a "committee" design, where dozens of users in various parts of the world have added components piecemeal. TAMU, on the other hand, has benefited both from experience gained from COPS, and from a smaller, more coordinated design and implementation team. As a result, TAMU tends to be easier to set up and use.

Of particular interest are the efforts the TAMU team have made to incorporate detection of flaws and obsolete system binaries. The current release of the TAMU package includes specific version information for AIX, HPUX, Linux, NeXT, SunOS and UNICOS versions of Unix. TAMU is also more aware of networked environments than is COPS. For example, TAMU will interrogate multiple sources of information (i.e. `/etc/passwd`, NIS, netinfo) to learn about user accounts.

6.4 TRIPWIRE

Tripwire was officially released on 2 November 1992 to Internet users.[6] Developed by Kim and Spafford, Tripwire as described in [34] and [35] is a passive Unix file system integrity checker. Tripwire works by comparing parts of the file system to previously saved file signature databases, alerting the system administrator to unexpected file system changes. These unexpected changes could indicate that intruders are active on the system, and that system security has been breached.[7]

TRIPWIRE PHILOSOPHY

Tripwire adopts a different approach to security than the other packages reviewed here. Instead of checking individual components for security weaknesses, Tripwire works by detecting unanticipated changes in system configuration files, system binaries and other tracked files.

The name "Tripwire" was coined when the developers realised that files with attractive names for intruders could be used as alerts. By watching such files for changes in content, access times, or other inode information, the administrator might be informed early of intrusive system users.

A number of design goals are identified in [35, p.3]:

- *Ease of use*

  In order for Tripwire to be widely adopted, the designers placed ease of use high on their list of design priorities. Ease of use is accomplished through a simple command line interface, and the separation of Tripwire data files into configuration files and signature database files.

[6] Interestingly enough, November 2nd is the anniversary of the Internet Worm.

[7] Tripwire version 1.1 was reviewed.

Tripwire configuration files specify which parts of the file system to include in the analysis. The files and directories to include are specified using a simple, but powerful, configuration language. Signature database files are generated from the configuration files, and are partially human readable.

Tripwire is designed to allow Signature databases to be easily maintained. Individual entries or ranges of entries may be updated without having to regenerate the entire database.

- *Automated runs*
  The designers of Tripwire intended that it be runnable by `cron`. This allows regular, unattended Tripwire comparisons to be run. Additionally, several different types of Tripwire runs can be performed automatically. For example, a superficial check could be run every couple of hours, while a more computationally intensive in-depth comparison could be run on a daily or weekly basis.

- *Reuse of configuration information*
  Tripwire is designed so that core parts of configuration files can be re-used by different machines. Configuration files may include preprocessor directives such as `@@include`, `@@ifdef`, `@@ifhost`, and `@@define`, allowing complex configurational files to be broken down into smaller, more reusable and scalable units.

  This is particularly useful in distributed environments. For instance, an organisation may have a number of networked machines running a standardised version of the Unix operating system, but with different applications and utilities installed on each one. A configuration file can be generated for standard operating system files to be tracked, and then `@@included` in customised per-machine files.

- *Easily examined output*
  Tripwire can potentially produce large amounts of output when files are changed or updated as a normal part of system operation. To address this, Tripwire supports "noise" reduction by allowing the administrator to specify changes to ignore. For example, the administrator can configure TripWire to ignore various types of change to specific files or directories. For example, log files can be flagged so that changes in size or signature are ignored.

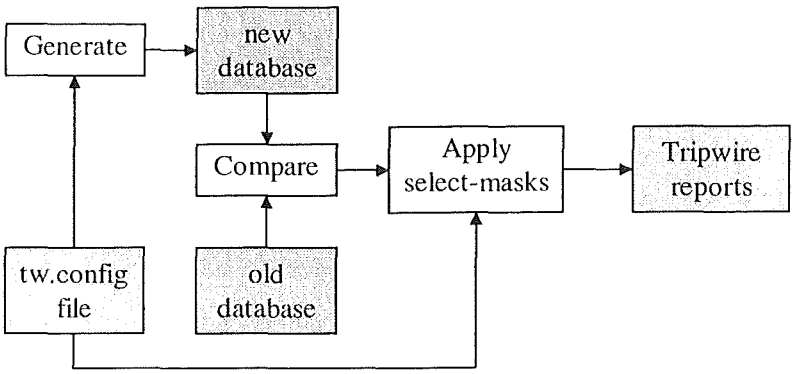


Figure 6 - 3 Tripwire high level operation (adapted from Kim et. al. [34, p.10])

- *Support for multiple signature regimes*

  Each file entry in a Tripwire database may have up to ten signatures recorded against it. This allows different signatures to be used according to local conditions or time constraints, or for additional assurance.

  For example, CRC checks could be used as the signature for frequently run quick checks. For more complete testing, MD-5 signatures, which take considerably longer to generate, could be used.

TRIPWIRE IMPLEMENTATION

Tripwire consists of one main executable file (`TripWire`) that performs all checking. An additional program called `siggen` is also provided so that individual file entries can be generated.

Figure 6 - 3 gives an overview of Tripwire's operation. The `tw.config` configuration file is used to decide which file system objects are to be included in the signature database. This is subsequently compared to an old database that has been previously generated, producing a list of changed files that are passed onto the select-mask mechanism for filtering.

The select-mask, specified in the `tw.config` file, allows the administrator to discard unimportant changes. These could include changes to log files, application data files, or to system configuration files that are dynamically changed by the operating system. Extraordinary changes are then reported to the administrator.

The tw.config file

The tw.config file is used to specify which file system objects are recorded in the signature database, and what select-masks should be applied when those files are found to have changed. Part of a tw.config file is given in Figure 6 - 4 on the next page.

```
# file/directory          selection-mask
/etc                      R                    #all files under /etc
□
@@ifhost huia.cosc.canterbury.ac.nz            #do this only for huia
/usr/local/secure         R                    #secure directory
@@endif                                        #
□
/etc/mtab                 L                    #dynamic files
/etc/motd                 L                    #
/etc/utmp                 L                    #
```

Figure 6 - 4 Excerpt from a sample tw.config file.

The first field in the `tw.config` file specifies the file or directory. By default, all files in specified directories are included in the signature database, although prefixes can be used to specify pruning of directory trees.

The selection mask is specified as the second field. The selection masks of `L` (log file) and `R` (read-only) above are templates for the underlying selection mask mechanism.

Directives such as `@@ifhost huia.cosc.canterbury.ac.nz` allow configuration files to be centrally managed, but still used for a variety of machines on a network of hosts. However, each machine must still (for obvious reasons) have its own signature database.

Tripwire modes

Tripwire may be run in one of four modes: Database Generation, Integrity Checking, Database Update, and Interactive Update. Database Generation mode allows a new signature database to be generated, while Integrity Checking mode performs the actual comparison. Databases may be updated either automatically (Database Update mode) or interactively (Interactive Update mode).

Signatures supported

Tripwire supports a variety of algorithms for generating file signatures. Performance varies according to the algorithm; in general, the more secure the algorithm, the longer a signature will take to generate. The relative merits of these algorithms are discussed in detail in [34].

Those included with the distribution include:

- *CRC-32 and CRC-16*

  These two CRC algorithms produce a 32 bit and 16 bit CRC value respectively. These algorithms have long been used as standard error detection codes. Both are fast to compute, and are useful for generating a speedy signature. Unfortunately, both algorithms (especially CRC-16) are relatively easy to spoof.

- *MD5 (MD4, MD2)*

  The MD5 (Message Digest Algorithm) generates very secure 128 bit signatures. Promoted by RSA Data Security Inc., this algorithm is considered state of the art. It builds on the speedier, but slightly less secure, MD4 algorithm. MD4 and MD2 (also by RSA Inc.) are also included in the Tripwire distribution.

- *SHA*

  The Secure Hash Algorithm (SHA) is the proposed NIST Digital Signature Standard. Approximately one half the speed of MD5, it has been noted that SHA is based on MD4, but with several key enhancements [36, 6.7].

- *Snefru*

  The Xerox Secure Hash Function (Snefru) was developed by Merkle at Xerox PARC. This algorithm is slower than MD5, but provides very strong authentication, and is the recommended MD5 backup.

TRIPWIRE COMMENT

As with any change detection package, the use of Tripwire is contingent on the configurational security of the tracked system being correct *before* Tripwire is installed. This can be achieved by re-installing the operating system from scratch, but this does not prevent difficulties with non-binary file configuration problems from occurring.

In addition, Tripwire can only detect security breaches after the system has been compromised. How soon after depends wholly on how often Tripwire is run, and this is a strong argument for installing Tripwire as a process that is run on a regular basis as a `cron` job.

Thus, Tripwire should be used in conjunction with other configurational audit tools, such as COPS or TAMU. This will ensure that the chances of an attacker successfully exploiting a configurational vulnerability are minimised, and that if they do, it will be detected.

6.5 OTHER TOOLS

Several other tools exist, but not included in this review. Some of these tools are commercial, while others are public domain. These tools summarised below:

NetWare: Novell's SECURITY.EXE.

This tool is distributed with every version of Novell NetWare. It is described in more detail in Appendix B.

NetWare: Kane Security Analyst (KSA)

KSA checks NetWare 3.1X servers for security loopholes in password strength, access control, user account restrictions, system monitoring, data integrity and confidentiality [2]. No detailed information was available about the checks performed by KSA.

Unix and VMS: The Security Profile Inspector (SPI)

Available for use within the U.S. Department of Energy, SPI is distributed by the Lawrence Livermore National Laboratory and CIAC/CERT. A passive tester which assesses password strength and detects security relevant changes, versions are available for both the VMS and the UNIX platforms [55].

VMS:Clyde Digital Security Toolkit

A commercial package for VMS, this checker assesses user access, capabilities and rights, object access, controls and protection, network security and VMS auditing facilities. Optional additions include baseline checking, which can used to compare the current setup of the target system to site or organisational requirements [55].

6.6 CHAPTER SUMMARY

This chapter has surveyed four configurational audit tools. As has been discussed, COPS and TAMU are both configurational audit tools that use similar static audit techniques. ISS, on the other hand is a much more aggressive active checker.

Tripwire, while not exactly being a configurational audit tool, is a good companion for the other tools reviewed, since it performs change detection functions that are important to post-attack recovery.

Part II

NetAudit - A tool for assessing NetWare LAN configurational vulnerability

CHAPTER 7

NETAUDIT INTRODUCTION

This chapter provides an overview of NetAudit, an automated configurational audit package for the NetWare 3.1x operating system. The objectives of NetAudit are presented, followed by a discussion of the NetAudit philosophical approach. This chapter concludes with a description of the NetAudit development process.

7.1 INTRODUCTION

Assessing the security setup of NetWare LAN file servers requires that the auditor collect and analyse a large amount of file server configuration information. Currently, very few tools are available to facilitate this process.

The NetAudit prototype, designed by the author, attempts to help alleviate some of the problems associated with configurational vulnerability analysis of NetWare systems. Using NetAudit, a skilled auditor can detect configurational vulnerability in NetWare systems, assess system conformance to organisationally defined security standards, and detect security relevant changes.

7.2 THE NETWARE SECURITY ENVIRONMENT

This section provides a brief overview some of the problems that NetWare administrators face in the process of administering security on these systems. For a more complete discussion of the NetWare operating system, see *Novell NetWare 386 - The Complete Reference* [71], *Novell's CNA Study Guide* [12], or *The Novell NetWare 3.11 Concepts Manual* [48]. For an overview and discussion of LAN and small system security concepts and issues, see Anderson [1], David [18], Salamone [67], and Seyfert *et al.* [70]. LAN virus infection issues are discussed in Wack *et al.* [77], as well as numerous other sources.

As noted in [81], NetWare operational security tends to suffer from a number of problems that can be traced to both technical and non-technical causes. These problems are restated here:

- *Network security is a complex issue.*

  NetWare security controls are complex, and as a consequence of this, the potential for unforseen interactions or incompatible usage is substantial. Accordingly, the role of the administrator is similarly complex. Not only are administrators concerned with system availability issues, but they must also deal with a multitude of network clients, all of whom may need different levels of access to applications, files and devices. Due to the complexities of the controls, keeping track of network objects can be difficult, especially without appropriate tools. Additionally, ensuring that controls are used consistently in multiple server networks is a constant problem.

- *LAN systems mutate.*

  NetWare systems seldom remain in a static configuration for long periods of time. Over the lifetime of the system, devices, applications, users, groups, and various other NetWare objects are added, removed or changed. A system may enforce reasonable security when it is first installed, but changes in the population and configuration of network objects can erode this security over time.

- *Some programmed threats thrive in NetWare environments.*

  Novell NetWare networks can be ideal places for viral infections to spread. Considering the typically insecure nature of workstations attaching to NetWare file servers, any slip in server security, especially where file system permissions are concerned, could lead to widespread virus attacks.

- *Network administrators may lack the appropriate skills.*

  Many smaller PC networks end up being administered by people who lack the appropriate skills to recognise that potential security vulnerabilities are present in the system they are responsible for, or who are not aware of the potential risks associated with configurational changes that they make to the system. For example, an inexperienced NetWare administrator may see nothing wrong with granting every user account SUPERVISOR equivalence.

- *Network security may not be promoted within the organisation.*

  The importance of network security may not be recognised by management or system administrators. This in turn manifests itself as a general lack of good security practice on the part of users. Day-to-day security policy usually defines standards for such issues as acceptable system usage, administration procedures, and system auditing. Vulnerabilities may occur due to a lack of such policy. For example, users may unwittingly create vulnerabilities within the network system by placing passwords in

batch files, leaving their unattended machines logged in, granting inappropriate file and directory rights to other users, or any number of other inadvisable, but common "problem" usage habits.

- *NetWare network auditors lack the appropriate tools.*
  The last problem that may lead to undetected NetWare configurational vulnerabilities is that there are very few, if any, effective tools available to detect them.

In chapter 3, complexity, user habits, administrator habits, a lack of security policy, maintenance, and introduced flaws were all identified as contributing factors to the overall level of technical vulnerability within a system. As illustrated by the problems described above, these general factors map easily onto the NetWare security environment.

Other life cycle issues discussed in chapter 3 can also be applied to NetWare. For instance, some vulnerabilities present in the NetWare operating system can be traced to design defects. The NetWare NCP packet spoofing attack described in Appendix B (see *Spoofing of LAN traffic*, Page 218), exists because of original design weaknesses in the way that NetWare performed packet authentication. Safeguards to address this vulnerability are available, but these must be retrofitted to the system, and when installed extract a significant performance penalty. However, without these safeguards, NetWare's entire security mechanism may easily be subverted.

Making an assessment of operational NetWare security will need to take the possibility of such an attack into account. For example, a configurational audit package could determine if the appropriate patches are installed and configured correctly.

Other NetWare vulnerabilities can be traced to implementation issues. For example, the cleartext password of a newly created bindery object, such as a print server, can be seen in the long identification field of that object (see *Object Reuse*, Page 209 in Appendix B). This is the result of an implementation oversight, where the network or application buffers used during the object creation process are not cleared after being used to set the password.

Clearly, NetWare suffers from the same general and lifecycle problems that beset other computing environments. The security of NetWare can thus potentially be improved by implementing a configurational audit tool that draws upon some of the philosophies and implementation methods that have been described previously in this thesis. With this in mind, the objectives of NetAudit are discussed in the next section.

7.3 NETAUDIT OBJECTIVES

Defining the objectives of the NetAudit tool involved drawing on three main sources. The first was *A Vulnerability Analysis of NetWare 3.11*, which is included in this thesis as Appendix B. This analysis systematically examines NetWare's security mechanisms, and the configurational issues that influence the observed strength of NetWare security. Appendix B highlights a number of possible weaknesses in the operational use of NetWare. These were to become important in the subsequent definition of the initial objectives of NetAudit.

Talking to NetWare administrators provided another source of ideas about the nature and particular features of the package. Administrators lamented the lack of tools for verifying the use of security mechanisms, the lack of information provided by NetWare utilities about the bindery database, and the difficulties of assessing file system access.

The final source of objectives was the analysis (in chapters 4, 5 and 6 of this thesis) of some existing configurational analysis tools. Some of the design and implementation philosophies from those packages influenced the final form of NetAudit. However, given the differences between Unix security mechanisms, and those implemented by NetWare, only certain types of checking methodologies were applicable to NetWare. The main objectives of NetAudit are as follows:

- *Vulnerabilities*

  Provide a means of detecting outright security vulnerabilities in the system being audited, and prioritising those vulnerabilities in order of seriousness. NetAudit is particularly concerned with errors that arise from incompatible usage with respect to security mechanism use, security administration oversights, and inattention to the overall security of file server controls.

- *Conformance*

  Provide a means of assuring that minimum levels of security are being maintained in the audited system, and that these levels conform to auditor, site, or organisationally defined standards. NetAudit provides a configurable baseline mechanism that is used to describe acceptable standards in the use of NetWare security controls. This mechanism is used to check conformance to those standards.

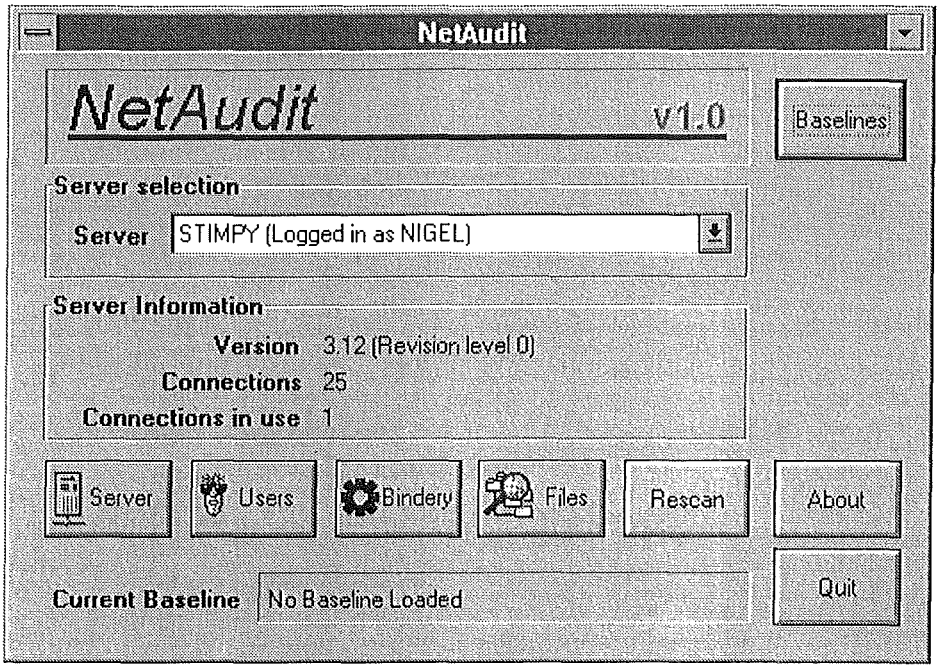


Figure 7 - 1 NetAudit main dialog

- *Change Detection*

  Provide a means of storing the security state of system objects so that changes in these objects can be detected. NetAudit currently detects changes in file system and user account settings.

- *Browsing*

  Provide a means of interactively browsing the security configuration of the target system. This helps the auditor visualise the security setup of that system by graphically representing certain relationships in security settings. NetAudit includes browsers for all major NetWare security relevant objects.

- *Reporting*

  Provide a means of creating reports that summarise or detail the security setup of the target file server, as well as individual objects or ranges of objects on that server. NetAudit provides a number of security and informational reports.

- *Ease of use*

  Provide a tool that is accessible to users, is intuitive, and that shields users as much as possible from underlying NetWare security complexities. NetAudit is a Microsoft Windows application that employs a graphical user interface to visually display NetWare security controls. Figure 7 - 1 shows the main NetAudit dialog, while Appendix D contains previews of other NetAudit interface components.

7.4 PHILOSOPHICAL ISSUES

The guiding philosophies of configurational audit tools were discussed in Chapter 5. These are important factors, as they influence the design and features set of the tool. The philosophical approaches adopted by NetAudit are as follows:

- *Passive checking.*
  NetAudit works by examining the static state of the target file server. No attempts are made to alter the state of the target system, nor does NetAudit attempt to remedy discovered configurational vulnerabilities.

- *Technical controls scope.*
  The configurational audit performed by NetAudit checks the technical controls that regulate bindery, userbase and file system security. NetAudit adopts a client-centric approach to checking NetWare controls, and performs only those checks that can be achieved through regular client access API's. Thus, server console and NLM security settings are not considered in NetAudit's analysis of file servers. Additionally, issues such as the security of individual LAN workstations are not considered.

- *Single vulnerability checker.*
  This version of NetAudit is a single vulnerability checker. That is, it does not attempt to assess the implications of multiple system vulnerabilities. However, insecure or non-conforming objects detected by NetAudit are ranked according to the number and seriousness of configurational errors they exhibit.

- *Reporting regime.*
  NetAudit incorporates magnitude indications in reports. For each message produced during an analysis, NetAudit indicates whether that message is a serious configurational error, or a warning. As mentioned in chapter 4, terse messages tend to favour attackers. The NetAudit user and bindery browsers include annotated explanations of discovered vulnerabilities. File system messages are considered reasonably self explanatory. NetAudit reports may also be exported to text files for further analysis.

- *Implementation language.*
  As this application is targeted at the Microsoft Windows graphical environment, and needs access to NetWare API's, a high level implementation language was required. Script-based and interpreted languages are not commonly available for the Windows environment, and those that are available (such as Visual Basic) suffer from severe performance drawbacks. Consequently, Borland C++ 3.1 was selected as the implementation language.

- *Tool Hazards*

  Because the majority of information required for NetWare security analysis is only available to SUPERVISOR or equivalent users, NetAudit requires access to the target file server as such an account. Due to the risks noted in Chapter 4, this is undesirable; however, NetWare 3.1x offers no other alternative (such as an "auditor" type user).

  Since it is intended to be distributed in executable form only, the chances of code subversion are minimal. However, care must be taken to ensure that NetAudit is never run from a machine infected by a virus. Because NetAudit is run with SUPERVISOR privileges, the danger from viral infections must be kept in mind.

NETAUDIT APPROACH

NetAudit's approach to testing configurational vulnerability is somewhat different to the other packages discussed in Chapter 6 of this thesis. The main differences in NetAudit's approach are its reliance on graphical browsing facilities, and the baseline checking methodology used to assess system vulnerability and conformance. These are discussed below.

Graphical facilities

As discussed on Page 101, NetAudit provides graphical browsers that allow the auditor to examine NetWare security mechanisms. While requiring substantial amounts of work to implement, these browsers promote the use of NetAudit as an investigative tool that can be used to interactively examine individual objects.

Baseline checking

Where other packages focus on specific system and technical vulnerabilities, NetAudit is based around the concept of *baseline* checking. Baselines are "ideal" object templates that are used as yardsticks against which objects residing on target servers measured. Objects that fail baseline tests may represent a risk to the security of the target server.[1]

NetAudit baselines have two objectives. First, by comparing file server objects to baseline settings, it is possible to detect where those objects vary from organisational security standards. Thus, baselines can be used to assure the auditor that security settings of objects on the target server are consistent with organisational or site policy. Second, a correctly

[1] The concept of a baseline is not new; baseline approaches have been used to assess conformance in fields as diverse as configuration management, financial performance, engineering, and medicine.

specified and applied baseline can detect object settings that may lead to a vulnerability in the security of the target system.

Currently, NetAudit supports separate baselines for user settings, bindery settings and file system settings. These baselines are fully configurable, allowing the security requirements of individual sites and organisations to be accommodated. Network auditors may have at their disposal a stock of pre-configured NetAudit baselines, each targeted at a different set of security needs. Pre-configured baselines may also be used as the basis of customised baselines for each site that the auditor assesses.

Change detection

In addition to baseline analysis, NetAudit can alert the auditor to security changes in server objects. By saving the security state of these objects, and then subsequently comparing saved objects to current versions, important state changes in object security can be detected. Where other packages concentrate on changes to the file system, NetAudit also allows the auditor to detect changes in the user population using the same techniques.

7.5 NETAUDIT DEVELOPMENT METHODOLOGY

The development of NetAudit was divided into four phases. Phase one involved learning as much as possible about NetWare security mechanisms. In phase two, a number of possible vulnerability exploitation scenarios were explored. Phase three was the design and implementation phase, while phase four was the testing and evaluation phase. These phases are outlined below:

1. *Review NetWare security controls*

   Phase one reviewed general NetWare LAN security. This analysis included intrinsic NetWare security controls, as well as more general LAN environment controls. As mentioned previously, this study appears as Appendix B to this thesis.

   Two main benefits were derived from the security review of NetWare controls: First, the analysis provided a comprehensive overview of the system objects that reside on NetWare servers. Second, the controls that protect those objects were identified.

2. *Identify potential vulnerability scenarios*

   In this phase, potential NetWare vulnerability exploitation scenarios were identified. Later, these scenarios helped form the basis of the checks that NetAudit was to perform. This phase drew upon the vulnerability analysis of NetWare given in

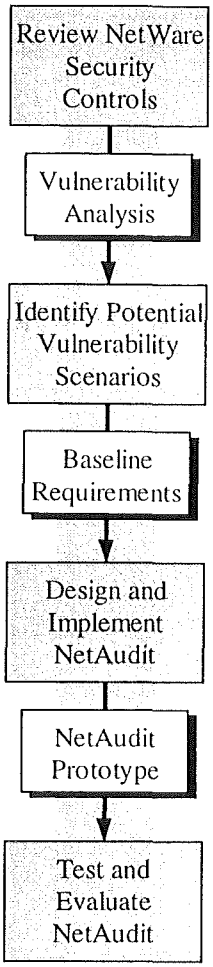
Appendix B, and used some of the techniques discussed in Appendix A. Phase two is discussed in more detail in chapter 8, *Determining NetAudit Requirements*.

3. *NetAudit design and implementation*

   During phase three, results from the previous phase, as well as the philosophies and implementation techniques discussed in chapter 5, were used to design and implement NetAudit. Chapter 9, *NetAudit Design and Implementation*, provides an overview of NetAudit's design.

4. *NetAudit testing and evaluation*

   The last phase of NetAudit's development had three objectives: The first was to assess how well vulnerability could be detected in NetWare systems using NetAudit. The second was to evaluate the value of the baseline approach; the third and final objective was to assess the effectiveness of NetAudit as security tool for NetWare. Chapter 10, *Using NetAudit*, describes a trial project conducted with NetAudit, and discusses how well the three objectives were met.



CHAPTER 8
DETERMINING NETAUDIT REQUIREMENTS

In order to arrive at a set of requirements for NetAudit, it was first necessary to determine how deficiencies in the controls of NetWare could lead to vulnerability. This chapter summarises the major components of NetWare technical security, and identifies potential vulnerability scenarios for each.

8.1 ANALYSIS SCOPE

The requirements analysis for NetAudit concentrates on those objects and controls that together regulate client access to the NetWare 3.11 and 3.12 operating system. Factors such as workstation security, physical security, network media security, and application security, although important, were not considered as part of the NetAudit requirements analysis.[1]

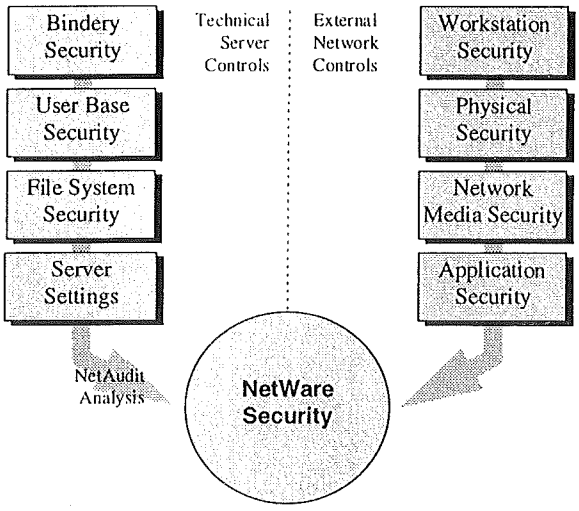


Figure 8 - 1 Scope of NetWare Security Model Analysis.

As shown by Figure 8 - 1, this analysis deals primarily with the technical controls that enforce NetWare bindery, user account, file system, and server security.

NETWARE SECURITY MECHANISMS

Many NetWare controls, and the objects that they protect, are internally managed and protected by the file server operating system. User and application access to such information is usually available only from standard NetWare utilities, or by requests made

[1] See Appendix B for a more wide-ranging discussion of these external factors.

directly to the operating system using the published set of NetWare API's (Application Programming Interfaces).

As mentioned in Chapter 5, this arrangement is at odds with mechanisms found in other operating systems for which configurational audit tools have previously been deployed. In NetWare, for example, user group membership is recorded in a structure private to the NetWare operating system. In contrast, Unix operating systems record group membership as an entry in a human readable text file. Both types of mechanism are protected by access controls (file system controls in the case of Unix, and bindery controls in the case of NetWare), but the net result is that more effort is required with NetWare in order to firstly understand the mechanism itself, and secondly identify possible sources of vulnerability within the mechanism.

In order to develop an automated configurational audit tool for NetWare, it is first necessary to build an understanding of how NetWare enforces security by identifying the crucial objects and controls of the operating system, and the interactions between them. From there, possible sources of configurational vulnerability may be identified. Once areas of potential vulnerability are known, the designer can assess what checks are necessary to detect these vulnerabilities

This chapter discusses the main NetWare security objects and controls. These include bindery objects, user accounts (a special form of bindery object), file system objects, and miscellaneous server settings. For each of object or control area, potential vulnerabilities are identified and explained.

8.2 BINDERY OBJECTS

The most basic NetWare security object is the bindery object. Bindery objects are an abstract class, from which subclasses of physical or logical NetWare objects (such as users, user groups, print queues, servers, and many other NetWare objects) are derived.

Instances of bindery objects are stored in the file server's *bindery*, a flat file database of all bindery objects managed by the server. Each NetWare file server on a LAN has a unique bindery, and bindery information is not actively shared between servers.[2] As will be

[2] Although a server's bindery may contain references to "advertising" bindery objects that regulate services on other servers or devices on the LAN.

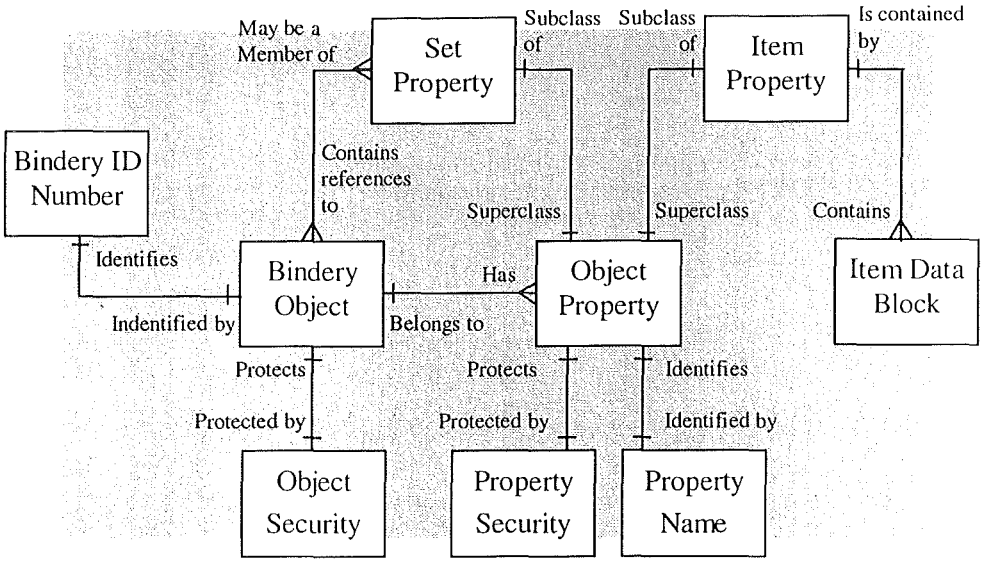


Figure 8 - 2 Bindery object entity relationship diagram.

discussed, the bindery database is the basis for NetWare login, access control, security equivalence, and group security.

Figure 8 - 2 shows, in entity relationship form, the general structure of bindery objects. As illustrated, a unique Bindery ID number is assigned to each bindery object for identification purposes. A textual bindery name is also specified, as well as an identifier indicating the object's type (user, user group, server, queue, etc).[3] A bindery object security field determines the level of bindery access required by other bindery objects to read or modify an object's bindery information. These levels are summarised in Table 8 - 1 on page 110.[4]

Each bindery object contains zero or more properties. Each of these properties is identified by a textual name, and is protected by a property security flag indicating the level of access required by other objects wishing to read or modify the property information. Property security levels are identical to those specified in Table 8 - 1 on page 110.

[3] The bindery name and object type data items are not shown in Figure 8 - 2.

[4] In the case of bindery objects, the object security flag controls what level of access is required to read, add or delete object properties.

Level	Read Access	Write Access
BS_ANY	Anybody can read the object, even non-authenticated objects	Anyone can modify the object, even non-authenticated objects.
BS_LOGGED	Only clients logged into the server can read the object.	Only clients logged into the server can modify the object.
BS_OBJECT	Only objects logged into the server with the object's name, type and password can read the object.	Only objects logged into the server with the object's name, type and password can modify the object.
BS_SUPER	Only SUPERVISOR or equivalent objects can read the object.	Only SUPERVISOR or equivalent objects can modify the object.
BS_BINDERY	Only the NetWare operating system can read the object.	Only the NetWare operating system can modify the object.

Table 8 - 1 Bindery object security levels.

Object properties are classified as either *set* or *item* properties. Set properties contain references to zero or more bindery objects, and are typically used to store group membership or access list data. For example, a bindery object seeking to act as a print queue operator requires membership in the target print queue object's Q_OPERATORS property. Such privileges are determined by the requesting object's bindery ID appearing in this property. If this is the case, then the queue operator access is granted; otherwise, it is denied.

Item properties, on the other hand, contain zero or more data segments. The type of data stored in these segments depends entirely on the role of the property. For example, the standard property IDENTIFICATION may store a long textual name for the object it belongs to. Alternatively, the LOGIN_CONTROL property stores a structure that defines various object access control settings.

Well formed standard bindery objects (such as users, user groups, print servers, etc.) are expected to contain certain properties. These properties vary according to object type, and according to the bindery relationships the object is part of. A badly formed bindery object may indicate that either an attack has taken place, or that the bindery database is corrupt. Table C - 1 (Appendix C, Page 227) summarises some of the more common bindery objects, and the expected properties of those objects.

Many other bindery object types exist; some defined by Novell, and others defined by third-party designers of NetWare services, applications, or devices. Some applications add additional properties to those that are standard for particular object types. For example, an

Control	Setting	Weakness	Exploitation Scenario
Bindery Object	Insufficient read protection	Bindery object information may be disclosed to unauthorised parties.	Attacker may browse bindery objects.
	Insufficient write protection	Bindery object information may be subject to unauthorised modification.	Attacker may change bindery object data. For example, the attacker may delete a property restricting access to some service.
Bindery Property	Insufficient read protection	Property information may be disclosed to unauthorised parties.	Attacker may browse bindery properties.
	Insufficient write protection	Property information may be subject to unauthorised modification.	Attacker may change property data. For instance, the attacker may add an unauthorised bindery object to the list

Table 8 - 2 Bindery object vulnerability scenarios

email application may associate an item property with user objects, the purpose of which is to track the last message that user accessed from the mail server.

ACCESS CONTROLS AND THE BINDERY

The bindery regulates access to a wide variety of physical or logical objects. For example, bindery object access to the file server is regulated by a password property. This property stores the password required before a client is allowed to log into the server as that object.

Objects may also store a list of other bindery objects that are entitled to access or manage the service that object represents. Print servers, for instance, specify a PS_OPERATORS property that defines which user objects or groups of users are entitled to perform administrative functions on behalf of the print server.

Resource accounting can also be achieved with bindery based security. In NetWare, an object may store an account balance, which is depleted every time that object uses a particular resource, such as disk space, connection time, or printing services. Once the account balance has decreased below a certain point, service requests may be denied to the object until the balance is increased.

The key to bindery security is the protection of the bindery database itself [5], and the controls that protect objects and their properties. If these controls are incorrectly configured, through modifications by applications, privileged users, or an attacker, the security of the server could potentially be compromised. For instance, if an attacker is

[5] Protection of bindery database files is discussed in more detail in *Important file system objects*, on page 122.

allowed to add an object that he or she controls to a list of privileged users of some other object, then the bindery security of that mechanism has effectively been circumvented.

Detecting vulnerability within the bindery thus involves checking that bindery object and property controls are sufficient to prevent unauthorised reading or modification of such objects. Some potential bindery object vulnerability scenarios are described in Table 8 - 2 on the previous page.

8.3 USER OBJECTS

Each user account on a NetWare file server is assigned a bindery object. This object is used to enforce user login security, for accounting purposes, and for user interaction with the file system. Detecting vulnerabilities in user accounts requires a different approach than that taken to assess bindery object security. In addition to the standard bindery security checks discussed above, the specialised nature of information held *within* user bindery objects warrants additional analysis. This section discusses that information.[6]

USER FILE SERVER LOGIN CONTROLS

The LOGIN_CONTROL property contains a structure that regulates user (or other object) server logins. This property represents one of the main repositories of login access control information for each user object.[7] Among the fields stored in this structure are:

- Account expiration information.
- Whether or not the account is disabled.
- Password expiration information.
- Grace login information.
- Password length restrictions.
- Concurrent connection restrictions.
- Login time restrictions.
- Bad login statistics.

Table 8 - 3 describes potential vulnerability scenarios that may arise as a result of configurational errors in the LOGIN_CONTROL structure. Checks of the information held in this structure should ensure that settings regulating password use and expiry, as well

[6] It is worth noting that assessing the security of any special purpose bindery object will involve an analysis of the contents of that object's properties, in addition to any checks of object and property security levels.

[7] The fields contained in the LOGIN_CONTROL structure are discussed in more detail in section B.3 of Appendix B.

Control	Setting	Weakness	Exploitation Scenario
Require Account Passwords	No.	Account may not be password protected.	Attacker logs in without having to undergo authentication process.
Allow User Password Changes	No.	User may not be able to specify own password.	Attacker learns of set password that user cannot change.
	Yes.	User specifies weak password.	Attacker guesses password.
Minimum Password Length.	None specified.	User is able to specify a short, and thus insecure password.	As above
Force Periodic Password Changes	No.	User is not required to periodically change passwords.	Attacker learns of old (but current) password.
Days Between Forced Password changes	Too many days specified.	User passwords do not expire often enough.	As above
Grace Logins before mandatory change	Too many grace logins.	Users can continue to use account password even after it has expired.	As above
Require Unique Passwords.	No.	Users can subvert forced password changes by re-specifying old password.	As above
Account Expiry Date	None specified	Account may become dormant.	Attacker exploits old or inactive accounts without arousing suspicion.
Limit Concurrent Connections	No limit.	Account could be used by two different people concurrently.	Attacker is able to log into account while it is in use.
Login Time Restrictions	No restrictions.	Account could be used at inappropriate times.	Attacker logs in after hours to reduce the risk of detection.
Login Station Restriction	No restrictions.	Account may be used from insecure or unknown locations.	Remote attackers log in to local servers from external (untrusted) networks.

Table 8 - 3 Login_Control settings vulnerability scenarios

as connection and time restrictions, conform to secure standards. This may be organisation dependant. For example, some organisations may require users to change passwords every month, while others may only force users to change passwords every six months.

USER PRIVILEGE SETTINGS

Users can gain additional privileges, either to NetWare devices and queues, or to the file system, in a variety of ways. The methods in which users gain additional privileges are discussed below.

- *The Supervisor object.*
  Every NetWare file server has a user special object with a bindery ID of `0x00000001`. This object, known as the file server SUPERVISOR account, is subject to very few restrictions; once logged in, the SUPERVISOR account has unrestricted access all parts of the file system and most parts of the bindery database.

  The SUPERVISOR bindery object also contains properties that store certain file server configuration information. This includes a list of server workgroup managers (discussed below), and various file server defaults.

- *Workgroup managers.*
  A workgroup manager is a normal user whom is also able to add users to the file server. Workgroup managers usually have limited SUPERVISOR control over users they have created.

- *Operators.*
  An operator is an account that has been granted special privileges to monitor and manage devices and objects such as print servers, print queues, and file servers. Operator privileges can be assigned to either groups or individual user objects.

- *Security equivalence.*
  As discussed in section B.4 (Appendix B, Page 195) , a user can enter into a security equivalence relationship with another user object. In this relationship, users can gain access to file system permissions that are granted directly to other users. Other privileges may also accrue from such a relationship. For instance, a user who is made security equivalent to the SUPERVISOR user is automatically granted the same rights as the SUPERVISOR to manage user settings and control devices.

- *Group membership.*
  Users are added to a particular NetWare group by making them security equivalent to the desired group's bindery object ID. Rights that are assigned to the group bindery object then become, via the equivalence mechanism, available to group members.

Table 8 - 4 on page 115 summarises potential vulnerability scenarios that may arise as a result of excess or inappropriate privileges being granted to a user account.

As illustrated, a configurational audit package needs to detect those users who are SUPERVISOR equivalent, those who are workgroup managers, and those whose group membership or security equivalence relationships are inconsistent with the security setup of

Control	Setting	Weakness	Exploitation Scenario
Security Equivalence	SUPERVISOR	Too many rights may be granted to user.	User may accidentally delete files important for the operation of the file server, or important to the organisation. Effectively no security for supervisor equivalent users.
	Non-user object	May cause erratic behaviour or incompatible usage problems.	May be used to crash the file server, or to grant undesirable access to server or queue file system objects.
	Too many granted.	Too many security equivalencies may cause unnecessary complexities determining actual user access to file system.	Unforseen interactions with the file system may be exploited by attacker. Attacker may be inadvertently be granted undesirable access to certain parts of the file system from security equivalence relationship.
	Inappropriate equivalence relationship	User may be granted an inappropriate security equivalence to another user.	Attacker may be granted security equivalence to a user that has access to sensitive data or applications.
Group Membership	Inappropriate group membership	User may be granted membership to a group that has file system access inappropriate for that user object.	User becomes a member of a group that has access to file system objects previously inaccessible to that user.
Workgroup Manager privilege	Granted to user object.	Too many rights may be granted to user.	Workgroup manager is a "sub" supervisor, that has SUPERVISOR rights over groups of other users.

Table 8 - 4 User privilege vulnerability scenarios.

the server. The server configuration should also be checked to ensure that operator privileges are not granted to inappropriate user accounts.

LOGIN SCRIPTS

Clients logging in as users using the standard NetWare login program automatically execute the commands that are contained in the system login script (see *Important file system objects,* on page 122). Once the commands in the system login script have been executed and the script exits, the user's login script is executed. User login scripts are stored in the `LOGIN` file located in the user's mail directory.

Because any user is able to create files in user mail directories, user login scripts should be checked to ensure that they exist [8], that they are sufficiently protected against unauthorised modifications, that they do not call unprotected batch files or programs, and that they do not contain hazardous commands that may compromise the account to which they belong.

[8] This vulnerability is discussed in more detail in Appendix B, Page 204.

Control	Setting	Weakness	Exploitation Scenario
Volume Restrictions	None specified	User may consume all disk space on volume, denying other users disk space.	Attacker continually generates files that consume disk space and directory entries, until no more space is available.
	Too much space granted	User may consume more than fair share of volume space, denying other users disk space.	Attacker generates files up to available limit.

Table 8 - 5 User file system restriction vulnerability scenarios.

FILE SYSTEM RESTRICTIONS

Restricting user object consumption of NetWare disk space may be an important facet of NetWare security. As summarised by Table 8 - 5, volume restrictions should be enforced to ensure that users do not unduly consume disk space, leading to denial of service problems for other users.

User file system access should also be checked to assess whether users are granted to parts of the NetWare file system to which they should not have access. Such checking should take the complexities of multiple group membership and security equivalencies into account, as well as the effects of the rights inherited from parent directories. This is discussed in more detail in the next section.

8.4 FILE SYSTEM OBJECTS

The bindery database and the file system act in unison to provide a discretionary access control mechanism for file system access. Assessing the configurational security of this mechanism is notoriously difficult, due to inherent complexities in its use. Consequently, side effects and inadvertent vulnerabilities are easily introduced into the file system, and difficult to detect. File system control mechanisms are described in this section.

FILE SYSTEM OBJECTS

The NetWare operating system has three subclasses of file system object. A *file object* is a basic file containing information or executable code. *Directories* are container objects, in which are placed other directories or files. A *volume* is a special type of directory object which forms the root directory of a logical NetWare disk partition. There are no container objects for NetWare volumes.

BINDERY AND FILE SYSTEM INTERACTION

Figure 8 - 3 on page 117 shows in entity relationship form the main interactions that occur between bindery objects and file system objects. As illustrated, these interactions occur in four places:

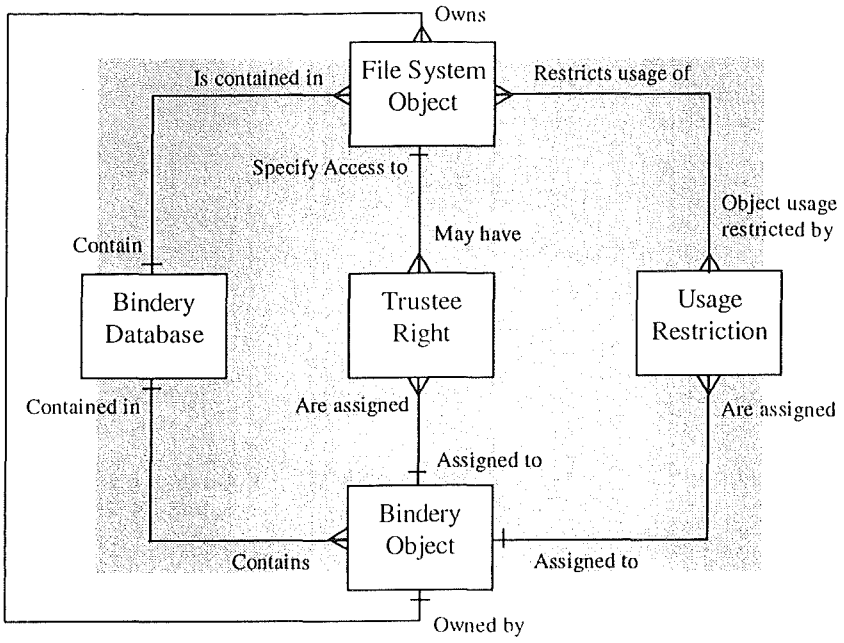


Figure 8 - 3 File system and object interaction overview.

- *The Bindery Database Files*

  The bindery database is stored in three files. Since the bindery database enforces much of NetWare's security measures, protecting these files from unauthorised deletion or modification is very important. This is discussed further in *Important file system objects* on Page 122.

- *Trustee Assignments*

  Trustee assignments may be attached to file system objects. These assignments specify individual or group bindery object access to the file or directory to which they are attached. *Calculating effective access,* on Page 118, discusses how these assignments are used to determine effective bindery object access to the file system.

- *File Object Ownership.*

  File system objects are owned by at most one bindery object (usually a user or a server). In some instances, a bindery object owning a file object may be deleted from the system, resulting in unowned files and directories. This is discussed further in *File owner checks*, on Page 121.

- *Usage Restrictions.*

  Usage restrictions are assigned to individual bindery objects for either a volume or a directory. This restriction limits the amount of space bindery objects are allowed to consume within the file system object for which the restriction is effective. Restrictions were discussed in the previous section.

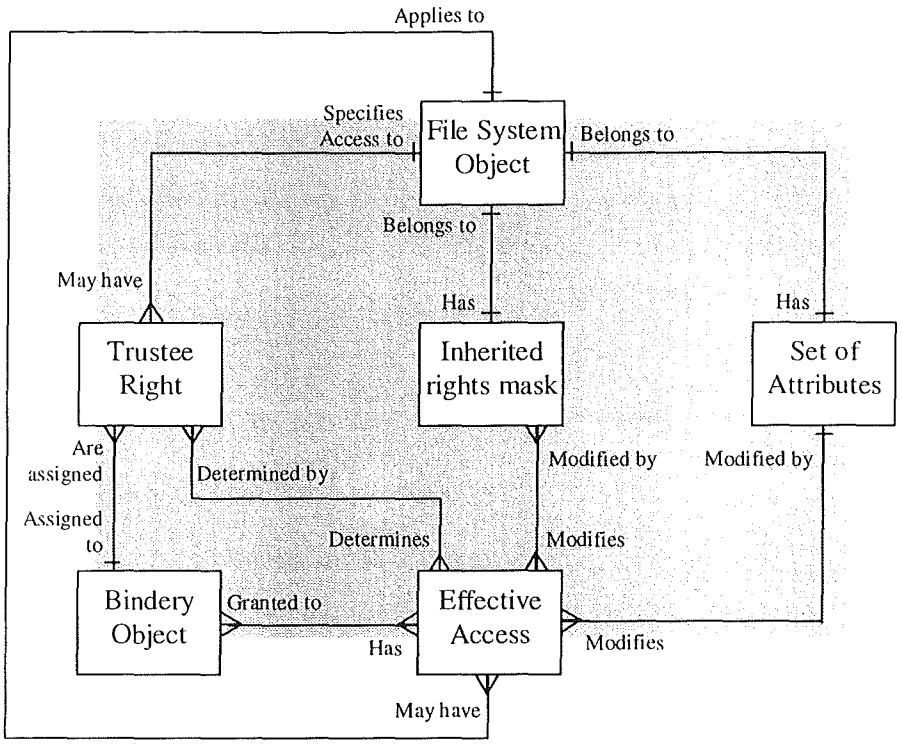


Figure 8 - 4 Relationships encountered during the process of calculating a bindery object's effective access to a file system object

In addition to the bindery and file system interactions introduced above, *File object attributes*, discussed on Page 120, influence the behaviour of file system objects regardless of the bindery object is requesting access. These attributes are purely a property of the file system object to which they are attached; however, they still have a significant impact on how bindery objects interact with the file system. Similarly, *inherited right masks* are attached to individual file system objects (usually directories), and modify the effective access that any non-supervisory bindery object would otherwise have to that file system object.

CALCULATING EFFECTIVE ACCESS

Figure 8 - 4 illustrates the entities and relationships involved in the calculation of a bindery object's effective access to a file system object. As shown, zero or more trustee right assignments are attached to each file system object. Each assignment describes the set of rights that a single bindery object, specified in the trustee assignment, has to that file system object. The bindery object specified in a trustee assignment can be either a single object, or a user group object. Where a user group is specified, the assigned trustee applies to each object that is a member of that group. The trustee rights available in NetWare 3.11 and 3.12 are described in Table B - 5, on Page 198 (Appendix B).

The process of calculating a bindery object's effective access to a file system object proceeds as follows:

1. *Trustee assignment exists for the bindery-file object pair.*
   If there is a file object trustee for the requesting bindery object, or any other bindery objects to which the requesting object is security equivalent, then the effective access granted to the requesting object (in this directory) is the logical OR of those trustee assignments. If any rights are effective for parent directories, then a logical AND of these parent rights with the target file system object's inherited rights mask is performed, and the results are logically OR'd with set of trustee rights already located for the target file system object. If a trustee assignment is located that contains the Supervisor right, or if this right is effective for the parent directory, then the access returned regardless of any other considerations, is always supervisory access.
2. *No trustee assignment exists for the bindery-file object pair.*
   If no immediate trustee assignment exists for the requesting bindery object, or for objects to which it is security equivalent, then the file system object's parent directory is searched for effective rights. If effective rights are found in the parent, then the effective rights returned are the logical AND of the effective parent rights with the inherited rights mask of the target file system object.

   This process proceeds recursively until the volume root is reached. If the search for trustee assignments completes with no suitable trustee assignments having been located, then the effective access returned is no access. If a supervisory level trustee assignment is located in any parent directory, then the effective access returned is always supervisory access.

Bearing this process in mind, Table 8 - 6 on the next page summarises potential trustee assignment vulnerabilities. Calculating effective access is a complex process. Not only do trustee assignments to the bindery object affect this access, but trustee assignments to security equivalent objects also have a bearing on the results. This, combined with the fact that trustee searches are performed from the directory in which access is requested back to the root of the volume, makes it very difficult to assess the results of trustee assignments and inherited rights.

This complexity can create configurational vulnerabilities, mainly due to unforseen interactions, or through misunderstandings of the file system security by those who administer it. This is especially true considering the lack of intrinsic NetWare tools

Control	Setting	Weakness	Exploitation Scenario
Trustee Right assignment	Granted in volume root directory	Because of the inherited rights mechanism, root directory rights may spread to subdirectories leading to unforseen interactions.	Attacker may access and/or change parts of the file system for which they have no authority.
	Granted in inappropriate directories	May allow user to compromise confidentiality or integrity of file system objects.	As above.
	Excessive Rights granted.	May allow user to compromise confidentiality or integrity of file system objects.	Attacker is granted write rights to a directory containing public binaries, and exploits this to insert a Trojan horse attack.
	Supervisory right granted	This right is unmaskable by inherited rights mask mechanism.	Attacker may be granted a S right at a high part of the directory tree, and may consequently use this to mount browsing attacks on the file system below that point.
	Access Right granted.	This right allows the holder to grant trustee rights (even ones they do not have) to themselves or other bindery objects to the file system object to which the right is granted.	Attacker grants access to inappropriate bindery object, resulting in unauthorised modification or disclosure of file system objects for which the right is granted.
	Inherited rights masks not used or incorrectly configured	File system object may not have been protected by an inherited rights mask limiting the rights objects inherit from parent container.	The attacker makes use of unintended flaws introduced because of this weakness. For example, attacker may be granted rights in the root directory of a volume, and there may be no inherited rights mask active for the SYS:SYSTEM volume. Rights granted to the attacker at the volume root are thus inherited in the SYS:SYSTEM volume.

Table 8 - 6 File system trustee rights vulnerability scenarios.

available for calculating and displaying a particular user's effective access to a specified part of the file system, or alternatively, a list of objects and the effective access that they have to a particular file or directory.

Thus, determining the presence of configurational vulnerabilities in file system trustee access controls will involve checking the effective access of each bindery object to interesting parts of the file system, and then deciding if that access represents a risk.

FILE OBJECT ATTRIBUTES

File object attributes may be applied to files or directories. These objects enforce an additional layer of access controls over and above those implemented by trustee assignments. For example, a read-only (Ro) file attribute could be applied to a file to

Control	Setting	Weakness	Exploitation Scenario
File and directory attributes	Rw setting on executable	Executable may be infected by a virus, or may be modified by attacker.	Attacker modifies executable program.
	Ro setting on executable, but M trustee right effective for users.	Executable attribute may be modified to Rw by the user, then infected by a virus or modified by an attacker.	As above
	X setting on executable, no rename inhibit setting.	Execute-Only (X) program can be renamed.	Execute-Only (X) program can be renamed, and another program of the same name substituted (companion style attack).
	H flag	File system object flagged as hidden	Attacker could hide files or directories containing stolen information or applications.

Table 8 - 7 File system file and directory attribute vulnerability scenarios.

prevent changes being made to it, regardless of the privileges of the bindery object requesting the change. Table B - 6 in Appendix B (Page 201) enumerates file object attributes, while Table B - 7 (Page 202) summarises likely interdependencies between file system trustee assignments and file system rights.

The most important role of file system objects from a security perspective is that they help limit the spread of viral infections. Of course, if a virus is NetWare aware, and is logged in as a user that is able to change the attributes of a target file, then such protection is of minimal value.[9]

Table 8 - 7 outlines attribute vulnerability scenarios. Checking file system object attributes is concerned with detecting executable and device files that are not flagged as read-only, locating hidden files and directories, and checking that other flags, such as rename inhibit and execute-only, are used where appropriate.

FILE OWNER CHECKS

Checking ownership of files has two aspects. First file owners should be checked to ensure that they are entitled to own files in a particular directory. For example, the `SYS:PUBLIC` directories should contain only files that are owned by either the file server itself, the SUPERVISOR account, or another recognised administrator. Second, a search for files that have no owner should be performed. Ownerless files usually belong to users or other bindery objects that have been removed from the system, and may represent a drain on

[9] This is also discussed in Appendix B, on Page 212.

Control	Setting	Weakness	Exploitation Scenario
File system object owner	Files owned by inappropriate owner according to file location.	Users with files in inappropriate areas of the file system	Attackers hide files in unexpected locations, or use file space that they are not entitled to. Attacker owns files in public directories.
	File or directory has no owner	No accountability for presence, content or intent of file. Could indicate unaccounted for file left over from deleted user. Consumes disk space.	Attacker could use unowned file to cover own tracks. Consumes disk space.

Table 8 - 8 File owner vulnerability scenarios.

resources if there are too many of them present on a system. Table 8 - 8, describes possible file owner vulnerability scenarios.

IMPORTANT FILE SYSTEM OBJECTS

From a security perspective, there are several important file system objects that are worthy of additional attention during a NetWare configurational audit. Table 8 - 9, on Page 123, summarises vulnerability exploitation scenarios of special file system objects, which are discussed below.

- *System directories.*

  There are several default directories, as identified on Page 203 of Appendix B, that are important to the operation and security of NetWare. Vulnerabilities may be introduced if the controls on these directories do not prevent unauthorised modifications from occurring. Checking the vulnerability of a NetWare server should include checks of which bindery objects are allowed modify access to these directories.

- *Bindery database files.*

  The bindery database is divided into three file system objects: `NET$PROP.SYS`, `NET$OBJ.SYS`, and `NET$VAL.SYS`. These files are located in the `SYS:SYSTEM` directory, and should be protected against modification by any user object. Additionally, old bindery files should be protected, as they may be stolen and analysed by an attacker.

- *System and user login scripts.*

  System and user login scripts, executed as users log in, should be checked to ensure that they exist. In addition, such files should be protected from unauthorised modification, and should be checked to ensure that they do not call unprotected batch files or programs.

FS Object(s)	Setting	Weakness	Exploitation Scenario
SYS:SYSTEM	Any access by normal user	This directory usually stores system files and configuration information that normal users should not have read or write access to. For example, some configuration files in this directory may contain passwords.	Attacker has write access and replaces system NLM with subversive version. Attacker finds password to remote console service in configuration file.
SYS:PUBLIC	Contents writable by normal user.	This directory stores executables and batch files that comprise the bulk of NetWare utilities. Write access by any user apart from the supervisor could facilitate the spread of viruses and worms. In addition, users may be able to replace legitimate executables with their own, or modify system login scripts.	Attacker replaces often used NetWare utility with subversive version. Attacker modifies system login script.
SYS:MAIL	Subdirectories writable by normal user.	These directories contain user login scripts. If normal users are allowed write access to other user mail directories (aside from the Create right), login scripts may be subverted.	Attacker modifies mail directory login script of another user.
SYS:LOGIN	Contents writable by any user.	The login directory usually stores the system login program, as well as some other utilities. Allowing writes to this directory will mean that a viral infection in one of those executables will infect every user that logs in. In addition, users could replace login executable with their own subverted version.	Attacker replaces LOGIN.EXE with their own version. Attacker infects LOGIN.EXE with virus.
Bindery database files	Writable by normal users.	The database files store the system bindery, and should be protected against unauthorised modification by any user.	Attacker deletes bindery files, thereby denying access to the server.
Application Directories	Writable by normal users.	Application directories should be protected against writes by normal users, but should allow read and file scan rights.	Attacker modifies or deletes applications.
Data Directories	Writable by unauthorised users.	In order to protect data integrity, shared data directories need protection from unauthorised modifications.	Attacker makes unauthorised "out of application" changes to data.

Table 8 - 9 Important file system vulnerability scenarios.

- *Protected directories.*

  Other directories may require protection from access or changes. For instance, there may be a directory on the file server containing confidential information. Checking

which bindery objects have access to that directory may be useful. In addition, there may be public directories containing applications, to which standard users should be allowed execute, but not modify, access.

- *System Binaries*
  System binary files should be checked to ensure that the version installed does not contain known flaws. As of writing, there are few acknowledged flaws in NetWare 3.11 or 3.12 binaries for which patches are available. Nevertheless, the utility of a configurational audit tool could be extended by including a component which detects when obsolete binaries are installed on the server.

8.5 MISCELLANEOUS SERVER CHECKS

This section discusses several miscellaneous checks of NetWare technical safeguards that do not fall into any of the categories already discussed.

PACKET SIGNATURE PROTECTION

The packet signature safeguards discussed in Section B.8 of Appendix B (Page 219) can significantly reduce the likelihood of attackers gaining unauthorised privileges to file server resources. It may be necessary to include a form of checking to assess whether or not these safeguards have been installed on target file server, and are operational.

CONSOLE SETTINGS

Some console settings can have an impact on the security status of the file server. For instance, the ALLOW_UNENCRYPTED_PASSWORDS variable, if set, allows clients to transmit passwords across the network in cleartext form. The risks of this practice are well know, and where possible, this console option should be set to OFF.

Another check that may be performed is whether the console has been secured. This removes DOS from the file server's local memory, disables the internal debugger, and restricts programs that are executed at the console to the SYS:SYSTEM directory. Securing the console in this fashion reduces the risk that an attacker with remote console access is able to run a hostile NLM from his or her own user directory.

A configurational audit tool may be able to check the Console settings to ensure that they are consistent with organisational security policy.

Control	Setting	Weakness	Exploitation Scenario
Detect Intruders	No	User accounts may be subjected to repeated password attacks.	Attacker subjects server to continuous stream of password guess attempts.
Incorrect Login Attempts	Too many specified.	Too many password guess attempts possible.	Attacker can check too many guesses in any one attack attempt.
Bad Login Count Retention Time	Too short a time period.	Bad login not "remembered" for long enough period of time.	Attacker can check many guesses over a longer period of time (waits between attempts, so as to not trip lockout).
Lock Account after Detection.	No.	Account is not locked even though there is enough evidence to surmise an attack is in progress.	Attacker can continue attacking server even though the attack has been detected.
	Yes	Attacker can exploit lockout to deny service to target account.	Attacker locks out SUPERVISOR and equivalent users to mask activity or prevent remedial action occurring.
Length of Account Lockout.	Too Short a time period.	User accounts may only be locked out for a short period of time before attacker re-commences attack.	Attacker locks out account, waits then starts again.

Table 8 - 10 Server intrusion detection settings vulnerability scenarios.

INTRUSION DETECTION SETTINGS

Intrusion detection settings are specified as a property of the file server bindery object. These need to be assessed by a configurational audit tool to ensure that they are active, and are configured to provide acceptable security. Table 8 - 10 describes possible intrusion detection vulnerability scenarios.

ACCOUNTING

The NetWare accounting service, while useful for tracking resources that user objects consume on the server, has more valuable security implications. With accounting active, the file server maintains more in-depth logs of user activity. A configurational audit tool should be able to inform the user when accounting is not active.

NLM SECURITY

NLM programs, as discussed in Appendix B, run at the maximum level of privilege supported by the operating system. Checking the protection surrounding NLM executables and configuration files should be an important issue.

In addition, where an NLM offers services or access to the file system to external network users, the configuration and access controls of these services should also be assessed for security. For example, an FTP NLM should be checked to ensure that only authorised

users are able to log in, and that the areas of the file system that the NLM can retrieve files from is limited.

Other examples of possible NLM vulnerability exist. For instance, placing the password for the RCONSOLE NLM module within server bootup files may represent an unacceptable risk for some organisations.

8.6 CHAPTER SUMMARY

This chapter has examined the main objects and controls that implement NetWare security mechanisms. As has been described, there are a number of possible areas of configurational vulnerability within the NetWare operating system. Thus, without some attention to configurational security issues, NetWare file servers may be vulnerable to attacks from skilled attackers, or from accidents made by legitimate users.

Two aspects of the analysis deserve comment: First, the configuration of NetWare security can be a complex process. Bindery object and file system object interactions can be extremely difficult to assess, because of the number of factors that influence object access to volumes, directories and files. Second, the way that administrators go about using the NetWare security mechanisms can have a marked effect on file server security. For instance, uncontrolled use of individual bindery object trustee assignments may lead to difficult to understand file system security structures.

Irrespective of external controls and management issues, the configuration of individual NetWare objects can either add to or detract from NetWare security. Nevertheless, it is important to remember that different organisations have different security needs, an aspect that configurational audit packages need to take into account.

The vulnerability scenarios presented in this chapter represent the output of this analysis process. These were further refined into baselines which were used in the implementation of NetAudit. Baselines are described in more detail in the next chapter.

CHAPTER 9
NETAUDIT DESIGN AND IMPLEMENTATION OVERVIEW

The previous chapter explained the process of deciding which checks were important for determining the extent of NetWare configurational vulnerability. This chapter describes how this information was used to design and implement NetAudit, a vulnerability analysis tool for the Novell NetWare 3.11 and 3.12 operating systems.

9.1 INTRODUCTION

On completing the analysis of NetWare objects and controls (discussed in Appendix B and the previous chapter), design and implementation of the NetAudit prototype could proceed. As discussed in chapter 7, NetAudit's analysis revolves around a baseline checking approach. Sections 9.2, 9.3 and 9.4 of this chapter describe how the analysis from the previous chapter was used to design NetAudit baselines assessing the respective areas of bindery, user and file system security. An introduction to the structure of the NetAudit package is provided in this section.

NETAUDIT STRUCTURE

The overall structure of NetAudit is shown in Figure 9 - 1 on Page 128. As shown, the tool is divided according to functional areas of security it addresses. The components of the tool are discussed in more detail below:

Bindery browser

This component provides an interactive browser that allows the auditor to view the contents of the bindery (see Figure D - 1 on page 229). This browser can display the security settings of bindery objects, as well as view the contents and security settings of object properties. The auditor uses this browser to launch bindery baseline analysis

sessions of single servers. The bindery browser provides facilities to view, print or save bindery security reports.

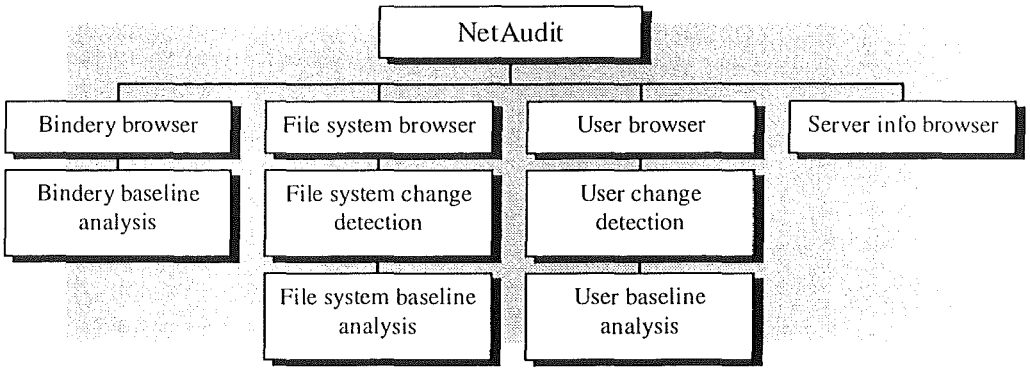


Figure 9 - 1 NetAudit program structure.

Userbase browser

This browser allows the auditor to examine the security settings of individual user accounts (see Figure D - 3 on Page 231). Facilities are provided for performing user account baseline conformance checks, and for saving user account information so that comparisons can be performed during subsequent audits. Reports produced from this browser include a user summary report, user details report, comparison reports, and security check reports.

File system browser

This browser provides a means of interactively viewing file system settings and information (see Figure D - 6 on Page 234). Facilities are also provided for baseline oriented analysis of file system objects, and for saving file system object state information for comparison purposes. Reports produced from this browser include volume usage and summary reports, disk space map reports, trustee reports, baseline check reports, and file system comparison reports.

Server information browser

The server information browser allows the auditor to collect general information about the target server (see Figure D - 7 on Page 235). Data available from this module includes server version information, accounting service status, intrusion detection status, and group and device information. In addition, configuration files and various log files can be viewed using this browser. This browser does not perform any baseline checks or comparisons; however, all information available from this browser can be saved as text files or printed.

Check item	Explanation
Object security levels.	Assess the security settings of individual bindery objects.
Standard property security levels.	Assess the security settings of standard bindery object properties.
Object equivalence to SUPERVISOR.	Flag objects that are security equivalent to the SUPERVISOR.
Property count threshold.	Flag objects that contain an excessive number of properties.
Notifiable property list.	Flag objects that contain a "notifiable" property.

Table 9 - 1 Bindery baseline summary.

9.2 BINDERY ASSESSMENT

Assessing bindery security requirements was discussed in *Access controls and the bindery* in Chapter 8 (Page 111). As stated, checking the security of the bindery revolves around ensuring that the protections of individual bindery objects and properties are correct, in relation to the type of object being checked. Table 9 - 1 summarises the tests that this baseline performs. Figure 9 - 2 on Page 130 shows the bindery baseline edit dialog.

The bindery baseline checks object and property security levels. For each type of bindery object, auditors can set the preferred read and modify security levels. Standard object settings, addressing common bindery object types, are included with the NetAudit distribution. Additional object type profiles may be easily added to the baseline. This allows NetAudit to accommodate third-party bindery object types.

The bindery baseline can also be configured to check the security of standard object properties. When an object is encountered that contains one of these properties, NetAudit checks the property security settings against a standard profile for that property type. Currently, NetAudit recognises twenty-four different properties.

In addition to object and property security checks, this baseline allows the auditor to locate objects that are security equivalent to the SUPERVISOR, objects that have an excessive number of properties, and objects that contain one of a set of auditor specified notifiable properties. This facility provides a means for the auditor to locate bindery objects that may have a special security significance. For example, the auditor may be interested in finding all objects containing a property that allows access to certain devices or services.

During analysis of the target file server bindery, NetAudit retrieves individual bindery objects from the server, and uses the baseline to decide which checks to perform. If the retrieved object is a type that has been included in the baseline object security settings, NetAudit compares the security of the retrieved object against the baseline specification.

Non-conforming objects are flagged. Once a bindery analysis has been run, the auditor is presented with a visual indication of which objects passed the baseline test, and which failed (see Figure D - 2 on Page 230). Failed bindery objects are ranked according to the number and seriousness of problems that were encountered during the analysis.

The NetAudit scoring system used is arbitrary, and is common to all baseline reports in NetAudit. Serious errors are awarded a score of 100 points, and marked in reports with `-ERR-`. Non serious or informational messages are awarded a score of 1, and marked with an `-INF-` flag. This mechanism allows the flagged objects to be ranked in order of seriousness. Objects that have serious errors, or a larger number of informational messages, end up with a higher score than those objects that have fewer, or less serious errors.

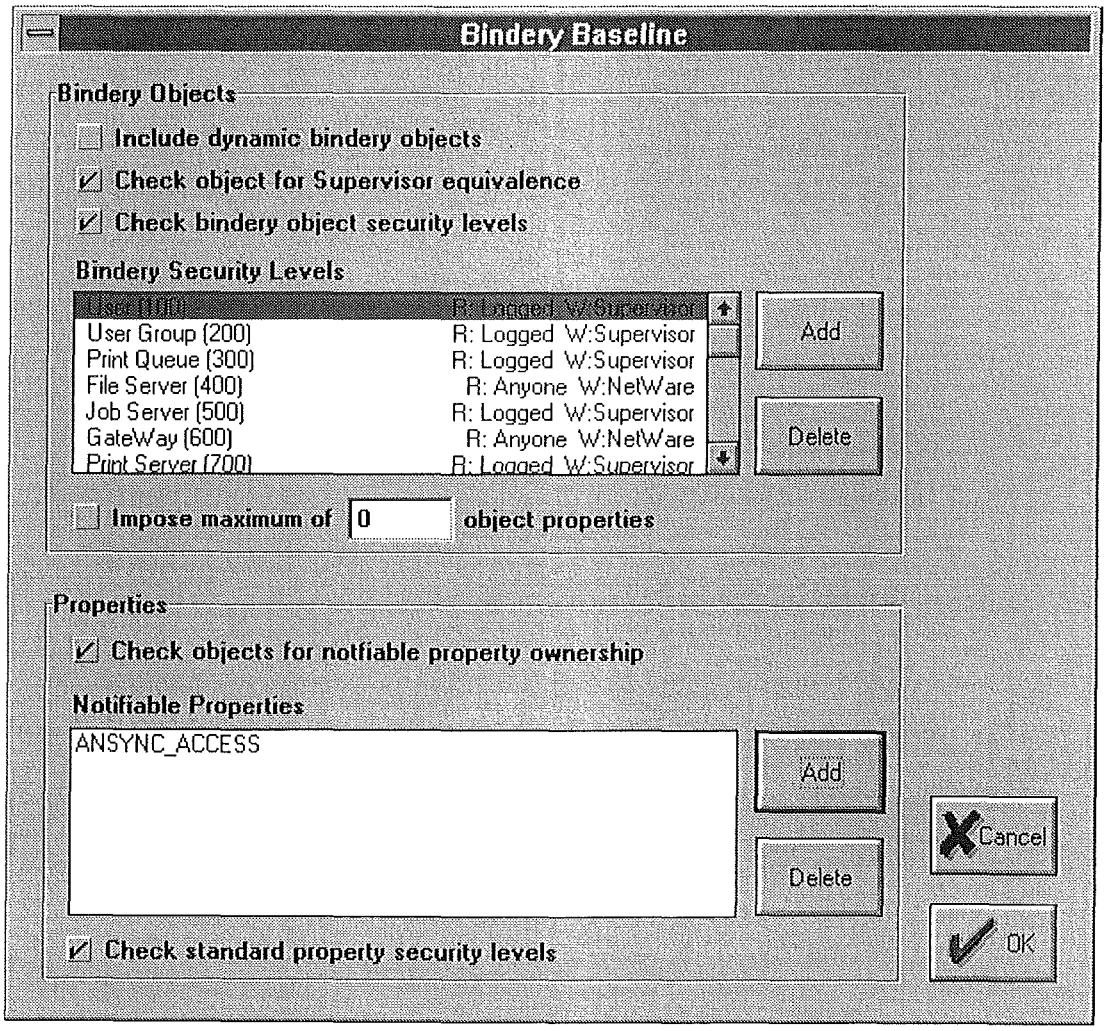


Figure 9 - 2 Bindery baseline editing dialog.

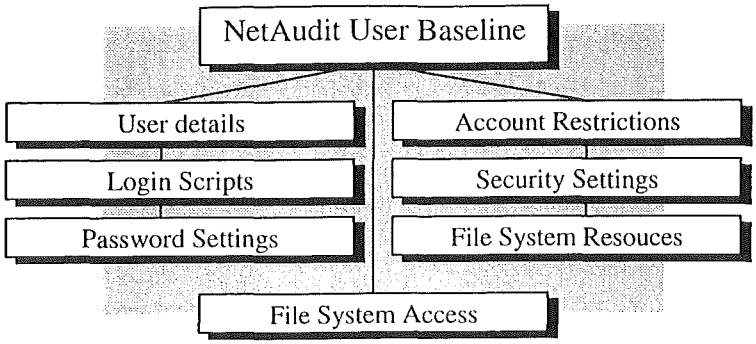


Figure 9 - 3 NetAudit user baseline checks.

9.3 USERBASE ASSESSMENT

Recall from the previous chapter that there are many security settings that influence individual user account security. As a result of the number of checks required, the user baseline is divided into seven more manageable sub-baselines. These are outlined in Figure 9 - 3, and are described below.

User details

The *user details* sub-baseline is summarised in Table 9 - 2. As illustrated, checks performed by this part of the user baseline are primarily aimed at detecting those accounts that are disabled, have not been used, or have no expiry date specified. In addition, the auditor may specify that accounts with no identification (full name) information are flagged.

This sub-baseline allows the auditor to detect the presence of an active GUEST account. At some sites, GUEST accounts are acceptable. However, good security practice suggests that guest accounts should be discouraged, due to the lack of accountability associated with their use.

User password settings

Vulnerability scenarios related to password controls were discussed in Chapter 8 (Table 8 - 3 on Page 113). These controls are contained in the user LOGIN_CONTROL bindery

Check item	Explanation
Require full account name.	Check accounts for full name.
Require account expiry date.	Flag accounts that have no specified expiry date.
Flag disabled account.	Flag accounts that have been disabled by intruder detection lockout, expiry or for any other reason.
Check inactive accounts.	Flag accounts that have not been used for a certain number of days.
Check dormant account.	Flag accounts that have been set up, but never used.
Allow GUEST account.	Check for the presence of the GUEST account.

Table 9 - 2 User details sub-baseline.

Check item	Explanation
Require account passwords	Flag accounts that have no mandatory password requirements.
Allow user password changes	Flag accounts that do not allow users to change their own passwords.
Require unique passwords	Flag accounts that allow users to re-use old passwords.
Require forced password changes	Flag accounts that are not subject to periodic forced password changes, or which have inappropriately long password lifetimes.
Allow grace logins.	Flag accounts allowing grace logins, or allowing too many grace logins.
Minimum password length.	Flag accounts that allow the user to specify a password that is too short.

Table 9 - 3 User password settings.

property, and regulate how users choose and change passwords. Table 9 - 3 details the *user password settings* sub-baseline used by NetAudit.

The auditor can configure this baseline to detect those users that have insecure password controls. These include users that are not required to password protect their accounts, users that are allowed to re-use old passwords, and users that are allowed to specify short passwords. Additionally, this sub-baseline can specify acceptable password lifetimes, and grace login usage. Accounts not meeting or exceeding these preferred guidelines are flagged during the baseline analysis.

A password cracker, although a useful addition to configurational audit, was not included in NetAudit prototype due to the difficulties of checking passwords via NetWare client API's. A third party NLM based password cracker, called SMARTPASS, is available to crack NetWare 3.11 and 3.12 passwords.[1]

Check item	Explanation
Allow concurrent logins	Flag accounts that allow more than one concurrent login session.
Require Node restrictions.	Flag accounts with no restrictions regarding which physical workstation nodes the account may be used from.
Require Time restrictions.	Flag accounts with no restrictions regarding the times that the account is able to be used.

Table 9 - 4 User account restrictions.

User account restrictions

User account restrictions vulnerability scenarios were discussed in Chapter 8 (Table 8 - 3 on Page 113). The *user account restrictions* sub-baseline is summarised in Table 9 - 4. Auditors can use this sub-baseline to detect user accounts that are allowed more than a specified number of concurrent logins, or that lack node or time restrictions. This version

[1] SMARTPASS is available from e.g. Software Inc [62].

Check item	Explanation
Check SUPERVISOR equiv.	Flag accounts that are SUPERVISOR equivalent.
Check Workgroup Manager.	Flag accounts that are workgroup managers.
Check equivalence to non-user objects	Flag accounts that are security equivalent to non-user objects.
Limit maximum number of equivalencies	Flag accounts with more than a specified number of security equivalencies (includes groups).
Check for equivalence to notifiable bindery objects.	Flag accounts that are security equivalent to objects nominated as sensitive by the auditor.

Table 9 - 5 User baseline security settings.

of NetAudit simply tests whether such node and time restrictions are present. No analysis is performed to assess the security impact of the times that logins are allowed, nor of the station from which these logins are permitted.

Security settings

The *security settings* sub-baseline is outlined in Table 9 - 5. The auditor can use this baseline to detect SUPERVISOR equivalent and workgroup manager users. In addition, the following checks are performed:

- *Check equivalence to non-user objects.*
  When specified, this check examines user accounts to determine the presence of security equivalence relationships with non-user or non-group objects. Any that are detected are flagged. As mentioned in Chapter 8 (Table 8 - 4 on Page 115), these may cause incompatible usage problems, or erratic behaviour on the part of the file server.

- *Limit maximum number of equivalencies.*
  This check detects user accounts that have in excess of a (baseline specified) number of equivalencies. Again, as noted in Table 8 - 4, too many equivalencies can lead to complexity and unforseen interaction; this baseline item allows the auditor to detect such complexity.

- *Check equivalence to notifiable bindery objects.*
  This check allows the auditor to specify bindery objects that are considered sensitive. A list of these bindery objects is specified by the auditor, and any user accounts that are found to be equivalent to these are flagged by NetAudit.

This version of NetAudit does not include checks of user operator privileges (discussed in *User privilege settings*, on Page 113 in the previous chapter). Such checks will be added to a future version of the package.

Check item	Explanation
Check for access in volume root.	Flag accounts with trustee assignments in the root directory of any volume.
Check for SUPERVISOR trustee.	Flag accounts with SUPERVISOR trustee rights.
Check for access in Tagged Dirs	Check effective access in nominated directories, and flag those users with rights in excess of those permitted.

Table 9 - 6 User file system access baseline.

User login scripts

User login script vulnerability scenarios were detailed in Chapter 8, on Page 115. There are two checks performed by the *user login script* sub-baseline: First, user accounts are checked to ensure that they at least have a login script file. Second, the file system protections of that file are checked for correctness.[2] Checks of the contents of user login scripts do not appear in this prototype of NetAudit, although adding such functionality would be reasonably straight-forward, and will be added in a future version.

User resource settings (volume restrictions)

Vulnerability scenarios related to volume restrictions were discussed in *File system restrictions* (Chapter 8, Page 116). The *user resource settings* sub-baseline allows the auditor to detect users who consume more than a certain percentage of any volume. In addition, NetAudit will optionally detect whether user accounts have volume usage restrictions, or if such restrictions are over-size (larger than the total volume size).

File system settings

The *file system settings* sub-baseline, shown in Table 9 - 6, allows the auditor to assess user trustee rights assignments. Two of the checks specified in Chapter 8 (Table 8 - 6 , Page 120) are performed by this sub-baseline: checking for trustees assigned in the root of a volume, and checking for SUPERVISOR level trustees.

NetAudit also allows the auditor to specify a list of *tagged* directories. A tagged directory is a directory for which a set of maximal effective rights are specified. For example, the auditor may specify that users are allowed a maximum of read and file-scan rights in the `SYS:PUBLIC` directory, and no rights at all in the `SYS:SYSTEM` directory. During an analysis run, NetAudit checks the effective access of each user account within tagged

[2] Alternatively, checking the permissions of user login scripts can be performed by setting up a tree profile on the `SYS:MAIL` directory specifying no write access for any user. For further discussion of file system profiles, see *File system assessment*, on Page 136.

directories. If this access exceeds the specified allowable rights, that account is flagged for attention by the auditor.

Using this mechanism, the auditor can designate areas of the file system as off limits or restricted access, and can detect those users who are assigned inappropriate rights in the directories specified. A file-system oriented check of trustee rights is also available. This is discussed further in *Directory tree profiles* on Page 137.

USER CHANGE DETECTION

In addition to the user conformance and vulnerability checks described above, NetAudit allows the auditor to record user configuration information for comparison purposes. Using these change detection features, the auditor can determine the effect of changes that have been made to both individual user accounts, and the general user population since the last configurational audit.

User comparison files are saved from the userbase browser. When directed to create a comparison file, NetAudit saves user configuration information for the currently displayed selection of user accounts. The comparison file is saved in binary format, and consists of the user account settings that are tracked by the user baseline.

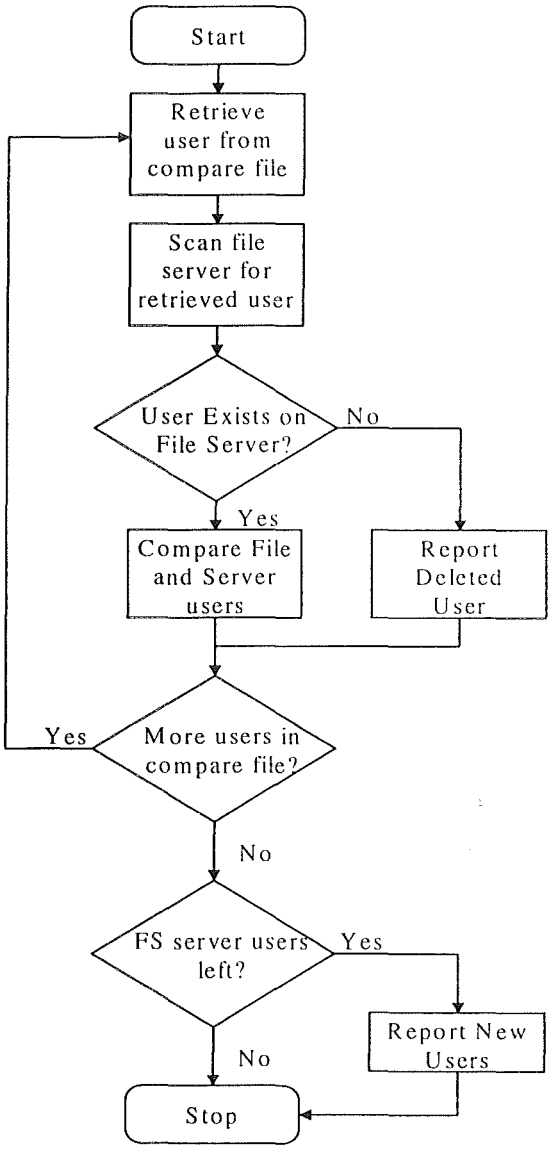


Figure 9 - 4 User comparison process.

Performing comparisons

Figure 9 - 4 illustrates the user comparison process. Once the auditor has specified a comparison file to use, NetAudit opens that file and uses it to determine a set of user accounts to compare on the target file server. For each user account found in the comparison file, NetAudit retrieves the same account from the target file server. In the event that the user account no longer exists, NetAudit reports

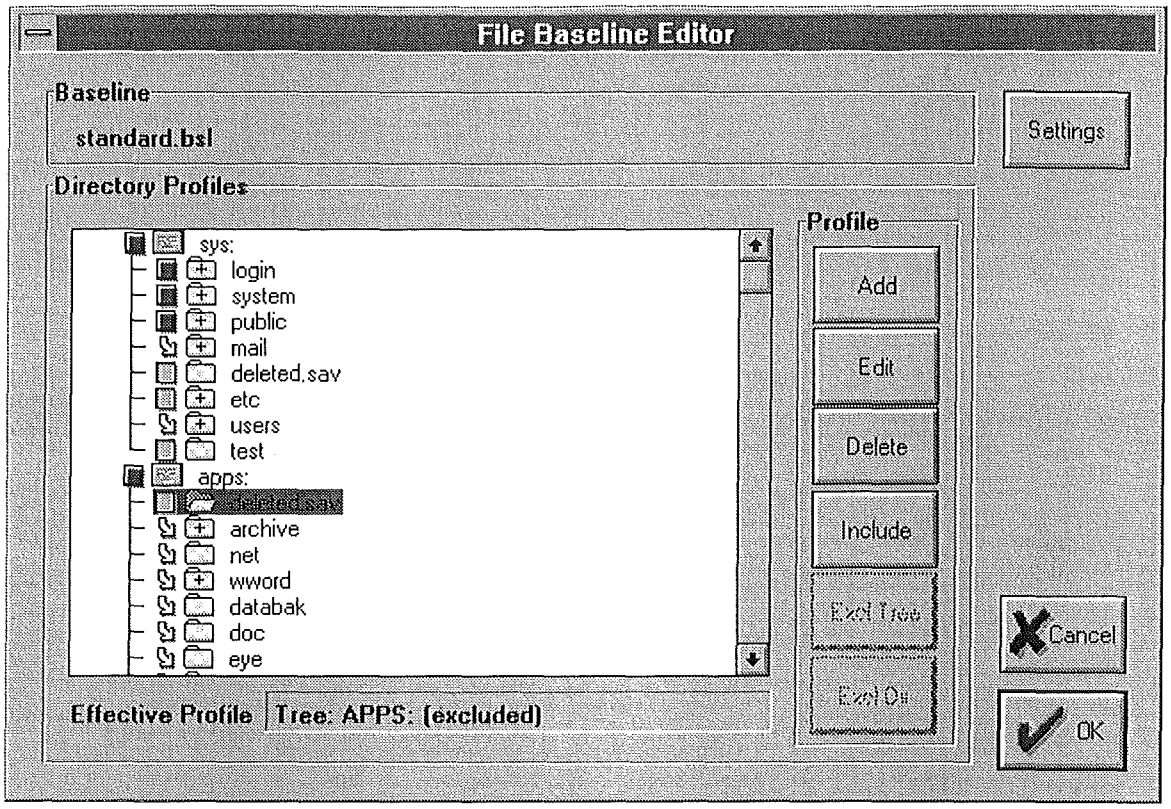


Figure 9 - 5 Main file baseline dialog showing profiled directories.

the account as a deleted user. If the user does exist, then NetAudit compares the current security state of the user to the saved version, reporting any changes that are found.

Once the comparison file is exhausted, any users left from the original selection are reported as users that have been added to the system since the last comparison was performed.

9.4 FILE SYSTEM ASSESSMENT

Checking NetWare file system object security and conformance is achieved by the NetAudit file system baseline. The main file system baseline edit dialog is shown in Figure 9 - 5. There are two main components of this baseline: A `SYS:` volume check, and a directory tree check. These are described in more detail here.

SYS: checks

As discussed in the previous chapter, there are a number of important objects (from a security perspective) on the `SYS:` volume. This check assesses the security of a variety of these objects. These checks (currently) include:

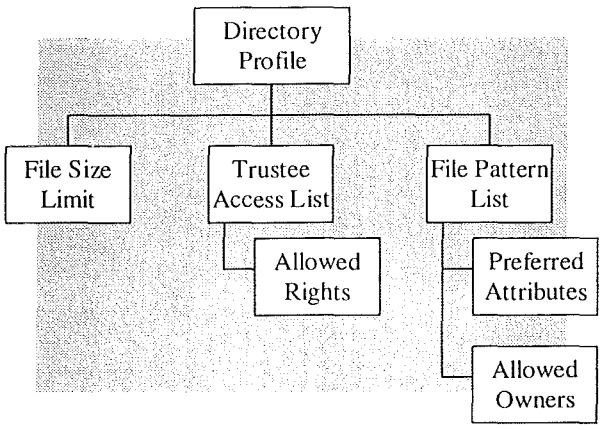


Figure 9 - 6 File profile structure.

- *Search for old bindery files*

  This check searches the `SYS:SYSTEM` directory for old bindery files. These files are generated by `BINDFIX.EXE`, and are often left unprotected.

- *Inherited rights masks on standard directories.*

  Security vulnerability scenarios related to incorrectly configured inherited rights masks were discussed in Chapter 8 (Table 8 - 6, on Page 120). The NetAudit `SYS:` volume check ensures that the inherited rights masks of the `SYS:SYSTEM`, `SYS:PUBLIC`, `SYS:MAIL` and `SYS:LOGIN` directories prevent rights granted in the volume root being inherited in these directories.

Directory tree profiles

Directory tree baselines, called *profiles*, are used by NetAudit to assess the security of file system objects. Profiles are applied either to single directories, or to entire directory trees. Figure 9 - 6 illustrates the structure of NetAudit file system profiles. Profiles are configured by the auditor to check file size limits, trustee access, file attributes, and file ownership. These items are described in more detail below:

- *File size limit.*

  This profile item allows the auditor to specify a size limit for files that are found within directories covered by the profile. Files exceeding this limit are flagged, allowing the auditor to locate large files in inappropriate parts of the file system.

- *Trustee access.*

  The trustee access profile item consists of a list of bindery objects and the maximum allowable rights of those objects within the profiled directory. Where necessary, a default profile can be specified that is applied to any object that has trustee rights in the

profiled directory. By specifying which bindery objects are allowed trustee assignments in particular directories, inappropriate trustee rights assignments can be detected.

- *File attributes and ownership.*
  This profile item consists of a list of file name patterns (such as `.EXE`, `.NLM`, `.DOC`, `DATA.TXT`, etc.) to which are attached specifications about the kinds of owners and attributes that are permissible for files matching these patterns. If a file is encountered in a profiled directory matching one of these patterns, the owner of the file is checked for membership in the permissible owners list of each file type. Additionally, the attributes of the file are checked to ensure that they conform to acceptable standards for that type of file.

  It is important to note that using file name patterns to determine file types is easily spoofed by simply changing the name of the file. Future versions of NetAudit may attempt to more accurately judge file type from an analysis of the file contents.

There are two types of profile available in NetAudit. Unless specifically excluded in a subdirectory, a *tree* profile applies to the directory it is specified in, as well any sub-trees located in that directory. *Directory* profiles apply only to the directory in which they are specified. In addition, the auditor can explicitly exclude from the analysis directories or trees that might otherwise be covered by a tree profile.

Tree and directory profiles are a flexible method of assessing the security of sub-sections of the file system. By specifying a tree profile in a volume root, for instance, the auditor can specify standards that are to apply to the entire volume. If there are directories or trees on that volume with special security requirements, additional profiles can be specified for those areas. These additional profiles take precedence over any active tree profiles specified in parent directories.

A typical profile tree setup for the `SYS:` volume is shown in Figure 9 - 7 on page 139. In this baseline, tree profiles have been set up in the `SYS:` volume root directory, the `SYS:LOGIN` directory, and the `SYS:SYSTEM` directory. The arrows indicate subdirectories where the tree profiles have been inherited, while the grey boxes indicate directory trees that have been excluded from the analysis.

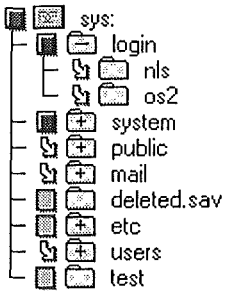


Figure 9 - 7 Typical SYS: volume file system profile setup (taken from NetAudit).

In this baseline, the auditor has specified a set of defaults in the root of the SYS: root directory that are to apply to most of the volume. These may include specifications for acceptable file attributes and owners for executables on the volume, file size limits, and a limit on use of the SUPERVISOR trustee right. The SYS:SYSTEM and SYS:LOGIN directories may have tighter restrictions specified, due to the sensitive nature of the files found in these directories.

Other file system checks

In addition to the checks described for the SYS: volume, and those carried out by directory and tree profiles, there are three other general tests that the auditor may specify. When active, these tests are performed for any directory that is not specifically excluded from analysis by the auditor. The checks are:

- *Hidden files and directories.*
  This option allows the auditor to locate any files or directories that may have been flagged as hidden (see *File object attributes*, on Page 120).
- *Files with no owner.*
  When specified, this option checks that files scanned during an analysis have a valid owner. Files that do not are flagged (see *File owner checks* on Page 121).
- *Owner not last updater.*
  This option checks that files scanned during an analysis have identical owner and updater fields. Those files that do not are flagged.

The directory profile mechanism described above can be configured by the auditor to address most of the vulnerability scenarios described in Table 8 - 6, Table 8 - 7, Table 8 - 8, and Table 8 - 9 in Chapter 8. Although a certain amount of effort may be required on the part of the auditor to specify the file system baseline, this mechanism is flexible enough to handle a wide range of site requirements.

FILE SYSTEM CHANGE DETECTION

NetAudit incorporates facilities for detecting changes within auditor nominated parts of the file system. By detecting alterations, the auditor can gain a better idea of how file system security has changed since the last configurational audit was performed. NetAudit can also determine where unexpected changes have occurred to file system objects.

NetAudit comparison files can be used to check server conformance. A comparison file containing file signatures of the standard contents of the `SYS:SYSTEM`, `SYS:LOGIN` and `SYS:PUBLIC` directories could be used by an auditor to check that appropriate system executables are installed on the target server.

Comparisons can detect the following changes:

- *Directory structure.*
  File system comparisons can detect added or deleted directories. These include hidden and system directories. NetAudit will also detect where files have been added to or deleted from a directory.
- *Directory trustee assignments.*
  Added, deleted or changed directory trustee assignments are detected. However, at this stage, the current prototype does not track file trustees.
- *File size changes.*
  Changes in the size of files, and the magnitude of those changes, are detected.
- *File object owner.*
  Changes to file or directory ownership are detected.
- *File object dates.*
  NetAudit can detect changes to creation and modify dates of files or directories.
- *File object attributes.*
  File and directory attribute changes can be detected during file system comparisons.
- *File contents.*
  NetAudit can optionally calculate a file signature that can be used to determine changes in the *contents* of individual files.

Saving comparison files

NetAudit saves file system comparison files in human readable text form. This allows the auditor to manually examine entries in the file. Where necessary, entries can be removed

from these files with no ill effects. It is also possible to update individual comparison file entries, although this prototype of NetAudit does not currently support this.

When directed to save a comparison file, NetAudit uses the current file system directory profiles to decide which portions of the file system to store. Any directory that has an active profile is saved as part of the comparison file. This allows the auditor to select only specific parts of the file system to track. NetAudit records directory, subdirectory and file information for each directory included in the comparison file.

File signatures

The auditor can specify whether signatures are stored as part of the comparison file. This prototype of NetAudit uses the RSA Data Security Inc. MD-5 message digest algorithm. This algorithm generates a very secure 128 bit signature based on file contents.[3] MD-5 was selected over CRC and other conventional hash codes because of the potential spoofability of these methods.

Performing comparisons

NetAudit uses the saved comparison file to decide which file system directories to include in the analysis. The comparison process is similar to that described for user comparisons in Figure 9 - 4 on Page 135. Directories and files that appear in the comparison file that do not appear on the file server are reported as deleted, while directories and files that appear in the profiled directories, but not the comparison file, are reported as new.

Changes to any of the tracked items mentioned above are also reported. Comparison reports are exportable as text files, allowing further filtering or analysis to be performed.

9.5 IMPLEMENTATION ENVIRONMENT

NetAudit was developed in the Borland C++ 3.1 environment, using the ObjectWindows and Application Frameworks class libraries included with this compiler.[4] The target platform for NetAudit was the Microsoft Windows platform, an operating system extender commonly found at sites running the NetWare operating system. There were several advantages of developing NetAudit as a Microsoft Windows 3.1 application:

---

[3] The code implementing MD-5 in NetAudit was borrowed from the Tripwire package (discussed in Chapter 6).

[4] See *Implementation language* on Page 102.

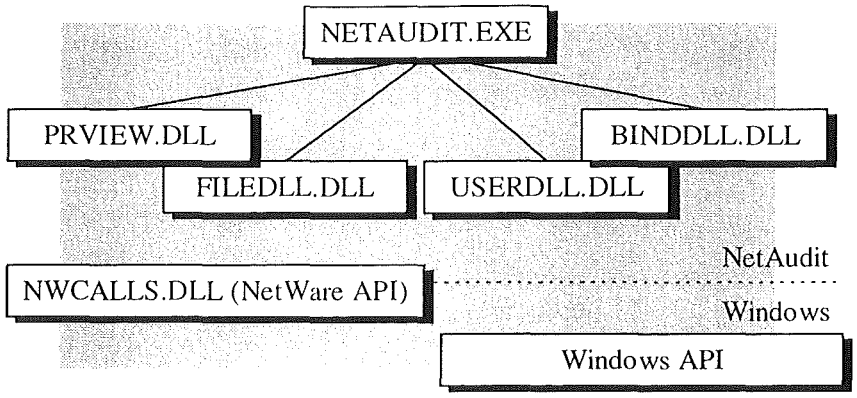


Figure 9 - 8 NetAudit executable structure.

- *User interface.*
  Windows makes it possible for developers to make extensive use of the graphical user interface metaphor. NetAudit uses makes use of graphics, fonts, and menus to present security information to the user.

- *Memory Management.*
  Windows provides a more sophisticated form of memory management than does MS-DOS. Large amounts of memory are handled automatically by Windows, easing the development process considerably.

- *Dynamic Link Libraries.*
  Windows supports the concept of Dynamic Link Libraries (DLL's). These modules export functions that are callable by other Windows applications. DLL's allow programs to be dividing into modules according to functionality, and facilitates more efficient use of Windows memory.

There were also drawbacks to the use of Windows as a platform. Because NetAudit is a graphical tool, a significant amount of work was required to design and debug the user interface component. In addition, documentation for some parts of the application programming interface for both Windows and for accessing Novell NetWare servers, were inadequate.

NETAUDIT EXECUTABLE STRUCTURE

Figure 9 - 8 illustrates the executable structure of the NetAudit application. As shown, NetAudit is divided into five distinct modules. These are explained below:

- *NETAUDIT.EXE*
  This is the main executable for NetAudit. This module implements the user interface for NetAudit, and contains code for the four main browsers of NetAudit. In addition,

this module includes baseline editing dialogs, server attachment and various other functions.

- *PRVIEW.DLL*
  This module contains code related to the saving, printing, and display of text reports from NetAudit. In addition, it exports a control to other NetAudit modules that facilitates viewing of text files of any size.
- *FILEDLL.DLL, USERDLL.DLL, and BINDDLL.DLL.*
  These modules implement data gathering, analysis and comparison functions for file system, userbase and bindery security respectively. There are no user interface elements in these modules, so they can be easily ported to other operating systems, such as Windows NT or OS/2.
- *NWCALLS.DLL*
  NetAudit communicates with NetWare using NWCALLS.DLL. This DLL, distributed by Novell, exports the NetWare client API.

9.6 CHAPTER SUMMARY

This chapter has examined the design and implementation of NetAudit, a prototype configurational vulnerability analysis tool for the NetWare operating system. The next chapter discusses how NetAudit was used in a trial analysis project. The results of this project, and an evaluation of NetAudit and its techniques, are presented.

CHAPTER 10
USING NETAUDIT

This chapter describes how NetAudit was used to assess NetWare system security at a production site. Results from this study are presented, along with an evaluation of the NetAudit package and approach.

10.1 TRIAL PROJECT

A NetAudit trial project was conducted at a major Christchurch academic institution. The campus-wide LAN of this organisation hosts a variety of shared computing resources, based around NetWare 2.2 and 3.11, IBM LAN Server, SCO Xenix, and Linux Unix. Several client operating systems are also used, including MS-DOS, OS/2 and Unix variants.

While a central authority is responsible for the overall administration and well-being of the LAN, individual departments are permitted to administer local systems connected to the network. NetWare 2.2 and 3.11 are commonly used by departments in a variety of roles, including staff, student, and support computing. With the cooperation of system administrators, NetAudit was used by the author to test several NetWare 3.11 systems at this site for configurational vulnerability.

The objectives of these tests were as follows:

- *Assess the value of conformance testing as a vulnerability detection technique.*
  A primary aim of this project was to assess how well NetWare configurational vulnerability could be detected by using NetAudit's baseline techniques. From this, several conclusions regarding the effectiveness of conformance testing as a means of detecting configurational vulnerability were made. Change detection was not included in this set of tests.
- *Assess the effectiveness of the NetAudit prototype.*
  It was important to assess how effective NetAudit was as a tool. Indeed, during the project, a number of areas where NetAudit could be improved were identified. These improvements will be included in future versions of NetAudit.

Server	Users	Function
A	247	Contained a mix of student and staff accounts. Most student accounts on this machine belonged to people enrolled in novice programming or computing courses.
B	324	Contained student and some staff accounts. User accounts for more advanced computing students were stored on this server.
C	532	This server held a large number of accounts used by students studying general subjects. In addition, a large number of guest account as well as many staff accounts resided on this server.
D	162	This server was used mainly by novice users, using MS-Windows applications.
E	87	This server was used as an experimental testbed for students learning about the NetWare operating system. Co-administered by students, this server contains only student accounts.

Table 10 - 1 Trial project server summary.

- *Assess the security of the target systems.*
  The final goal of the trial project, and the overall objective of NetAudit, was to assess the security of the target systems. Analysis of data from NetAudit reports revealed a number of interesting observations about the security setup of the tested systems.

Target server selections

From a server population of about twenty file servers, five NetWare 3.11 servers, controlled by a single department within the organisation, were selected as targets. These systems accommodated a variety of needs, and supported a disparate mix of users. The servers selected are summarised in the Table 10 - 1. Of these servers, four were administered by department personnel. Server E was administered in conjunction with some of the organisation's students.

BASELINE SETUP

Initially, a suitable baseline setup was discussed with administrators at the site. This baseline embodied the site practices that were active at the time; however, the baseline that was defined by no means represented most secure settings that could have been employed for conformance testing. Rather, the resulting baseline was suitable *relative to the security needs of that particular site*. Baselines used during the trial project were identical for each server. This allowed configurational conformance to be subsequently compared between systems.

Of particular importance to this analysis was the large number of guest accounts that were active on the systems tested. Guest accounts, while being inherently risky, are sometimes the only way to overcome technical security problems in NetWare environments consisting

of multiple servers. These guest accounts were to have an impact on the results of the trial project.

For each file server, a set of standard baseline runs were performed for bindery, userbase and file system security. NetAudit reports generated from these runs were collected during the project test sessions, and immediately obvious security problems that were detected were pointed out to the administrators of the affected systems. Reports produced by NetAudit were later analysed by the author after the NetAudit session; tools used during this additional analysis included standard text processing tools such as grep, awk, sort, wc and uniq.

BINDERY ASSESSMENT

An initial scan of the target servers revealed that most objects in the binderies of these systems were standard types. Thus, the baseline selected for bindery checking addressed standard object and property security levels. Apart from SUPERVISOR equivalent objects, no other notifiable bindery objects were specified.

Bindery results

Applying the bindery baseline to each of the servers in the analysis revealed no major problems. There was no evidence of tampering with any of the binderies tested. All objects that were flagged as SUPERVISOR equivalent were administrative user objects, with legitimate requirements for such access.

Several print queue and server objects were flagged for inadequate object or property read security. In the case of flagged print servers, a bindery object read security level of ANYONE was specified. This turned out to be a requirement of a printer resource accounting package installed and in use at the site. Some print queue objects were flagged for inadequate Q_DIRECTORY property read security settings, although this read access did not present a threat to the security of the tested servers.

USERBASE ASSESSMENT

The user baseline was a somewhat simplified version of the standard baseline distributed with NetAudit. Because of the particular security arrangements at this site, several items that would normally be included in the user baseline were omitted. The baseline items selected to take part in the analysis, and those omitted, are discussed below:

- *Account inactivity*

  Discussions with administrators revealed that accounts were considered inactive if they were not used for more than 14 days. This figure was used as the baseline specification for account inactivity.

- *Dormant accounts.*

  Dormant accounts are a problem at the site, due to "batch" setup of default class accounts, some of which are inevitably not used. Detecting these accounts is important, as they can potentially be misused by users to gain access to additional printing and disk resources to which they are not entitled.

- *Disabled accounts.*

  This baseline item checked for accounts that had been disabled. Most user accounts tested lacked expiry dates, as these were not in widespread use at the site. Accounts were generally only disabled through the intruder detection lockout feature (where it was active), or as a result of administrator actions. However, during the analysis a small number of accounts tested were discovered to have expiry dates; some of these had in fact become disabled through account expiry.

- *Login scripts.*

  The existence of login scripts is always an important NetWare vulnerability factor. Accounts without login scripts were flagged.

- *Password settings.*

  As per site practice, the baseline specified that users were allowed to re-select previously used passwords, and that a minimum password length of five characters was required. However, there were no site guidelines regarding password expiry (no forced password changes or grace logins). Consequently, these settings were not included in the analysis.

- *Account restrictions*

  The only account restriction important to this analysis was a limit of two concurrent logins per account. Time and node restrictions were not used at this site, and were left out of the analysis.

- *Account security settings.*

  Accounts were checked for equivalence to SUPERVISOR, workgroup manager privileges, and equivalence to non-user objects.

Server	A	B	C	D	E
Total number of accounts	247	324	532	162	87
Potentially vulnerable accounts	112	320	267	47	87
Tag violations	28.8	98.1	0.0	10.5	58.6
Dormant	9.3	35.2	33.8	5.6	12.6
Inactive 14 days or more	9.7	2.8	10.0	7.4	85.1
No password required	3.2	8.3	8.3	11.1	31.0
NULL password	3.2	7.4	4.5	11.7	31.0
Less than min password length	1.6	7.4	7.7	2.5	32.2
Disabled	0.4	0.0	7.3	0.0	27.6
S trustee assignments	0.4	0.0	0.0	0.0	48.3
No login script	1.6	0.9	0.0	1.2	0.0

Table 10 - 2 Trial project results from userbase analysis.[2]

- *User resource settings.*
  User accounts were checked for a greater than 10 percent usage share of any volume on the target server. The site had experienced problems in the past with users monopolising disk space. Since that time, volume restrictions limiting user disk space consumption had been enforced.

- *User file system interaction.*
  The ability of users to write to standard NetWare filesystem components was assessed as part of the user baseline. Specifically, rights in the `SYS:SYSTEM` (none), `SYS:MAIL` (Create), `SYS:PUBLIC` (Read and Filescan) and `SYS:LOGIN` (None) directories were assessed.[1] This baseline also checked for root and Supervisory trustee assignments.

Userbase results

The main results from userbase assessments of the test servers are shown in Table 10 - 2, and are summarised in graph form in Figure 10 - 1 on Page 150. For each vulnerability category shown in Table 10 - 2, the percentage of the total user accounts on each server with that particular vulnerability are shown. Legitimate administrative accounts (i.e. SUPERVISOR and equivalent users) were removed from the tagged directory and Supervisory trustee assignment analysis figures, as they automatically have access to the

[1] Trustee assignment checks in other directories were performed as part of the file system assessment, discussed on Page 152.

[2] Accounts with more than one tag violation or S trustee assignment were only counted once.

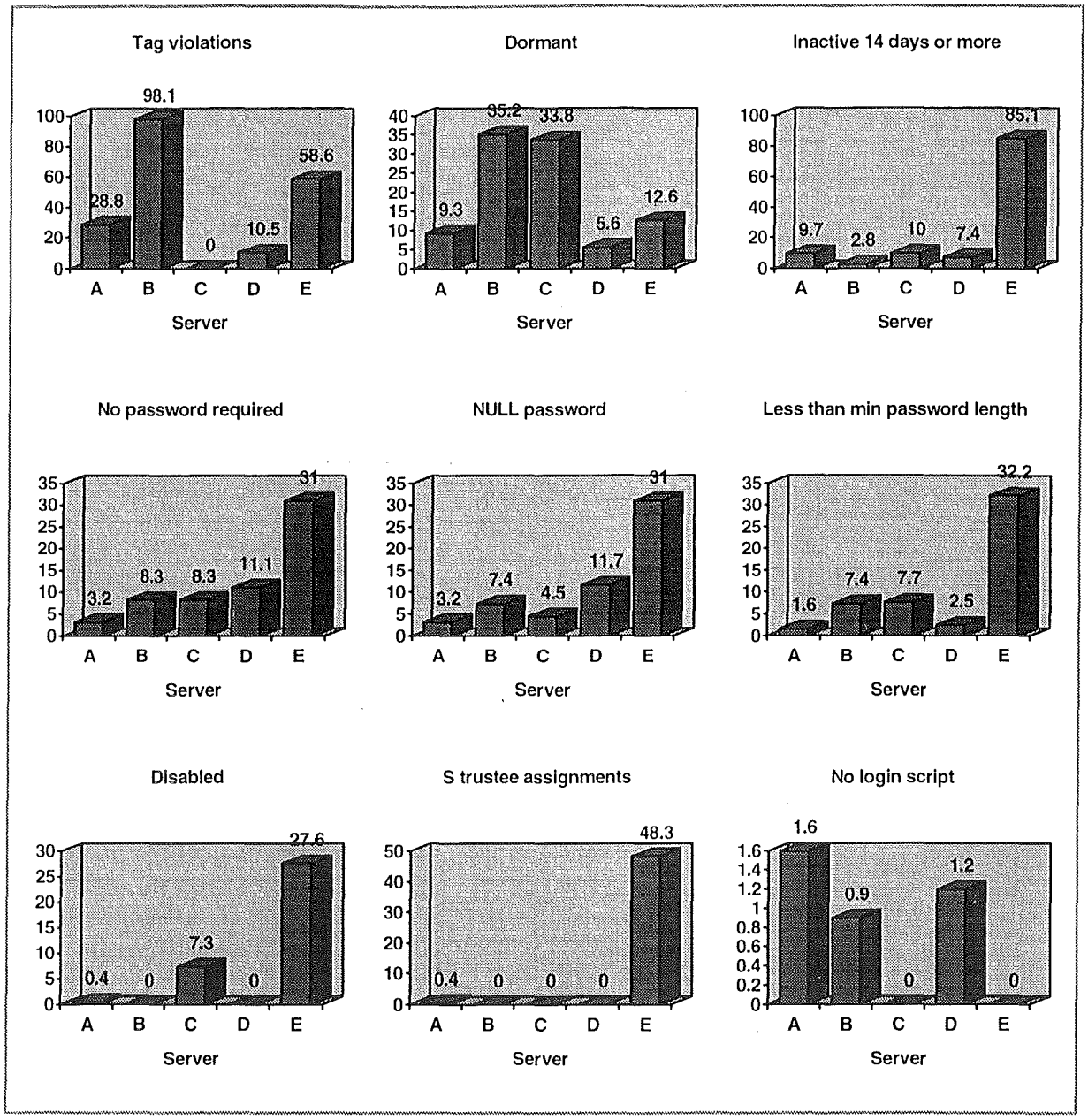


Figure 10 - 1 Summary of NetAudit userbase analysis. These graphs compare the percentages of accounts with the various categories of vulnerability that were found during the analysis.

entire file system. As these figures suggest, there were marked configurational variations between the servers taking part in the test project.

The largest percentage of potential vulnerabilities shown in Table 10 - 2 are those associated with directory trustee tag violations. There was an inordinately high percentage of tag violations observed on server B. Examination of that server revealed that the EVERYONE group was accidentally granted Write trustee rights to the `SYS:MAIL`

directory, and Read/Filescan rights to the SYS:LOGIN directory. Since most user accounts on NetWare servers are members of the EVERYONE group, nearly every account on server B was flagged as potentially vulnerable. A similar problem existed on servers D and A, although in those cases different groups, containing less members, were assigned write access to mail directories. Site administrators were unaware that the writable mail directory vulnerability was present.[3]

Several other tag violations were noticed. On servers A and B, two staff member user accounts were granted Write access to the PUBLIC directory. Checking revealed that these assignments allowed the staff members involved to update login notices located in that directory. The accounts belonging to these staff members were otherwise secure. The large number of tag violations on server E were the result of users being individually assigned Supervisory rights to their own mail directories.

The next most common problem encountered on the test servers were dormant accounts. Significant proportions of the total user account population on Servers B and C had never been used; subsequent investigation revealed that on these servers, a number of new accounts had recently been added in preparation for the coming term. Dormant accounts on the other servers were also of interest. Again, accounts had been set up for classes, but were never used. Some of the detected accounts had been present on the system for considerable periods of time, and were protected by default passwords. The fact that expiry dates were not used on these servers compounded the risks associated with these accounts.[4]

A number of inactive accounts were located on the test servers. Given that the baseline specification for inactive accounts was only 14 days, the number of accounts flagged was surprisingly small compared to the number of users tested. One exception was server E, where just over 85 percent of the user accounts were detected as inactive. This is presumably indicative of the experimental nature of that server.

A number of accounts on each server were found to have NULL passwords. The main factor in this figure was site reliance on guest accounts, for which passwords were not

---

[3] In this case, granting Write access to users in the SYS:MAIL directory allows those users to open and write (but not read) an existing file. For example, an attacker could, with this rights assignment active, change the SUPERVISOR login script. Normally, only the Create right is granted in this directory tree.

[4] Although server C had previously implemented user account expiry dates, resulting in a relatively high number of disabled accounts on that server.

used. Nevertheless, a number of staff and student accounts on servers A and B also lacked passwords. A noticeable (but not surprising) relationship appears to exist between the number of accounts allowing users to specify passwords of less than 5 characters, the number of accounts not requiring users to specify a password, and the number of accounts with NULL passwords.

Except for server E, Supervisor trustee assignments were not a common feature of the servers tested. Again, the high number of these assignments, combined with the relatively large number of (manually) disabled accounts on this server indicate its experimental nature.

Accounts with no login script were rare, but there were still a total of 9 (from 1352 accounts tested) which lacked such scripts. Fortunately, none of these accounts were privileged, or belonged to staff members.

Other user security issues were also addressed during the analysis. The incidence of SUPERVISOR equivalence encountered was low, with typically two to three such accounts on each server. Workgroup managers were uncommon, and there were no user account managers on any of the servers tested. Resource use did not appear to be a problem, with the only disk intensive account detected belonging to one of the SUPERVISOR equivalent administrators. All accounts tested had volume space restrictions enforced.

FILE SYSTEM ASSESSMENT

A standard baseline was devised for file system analysis of the test servers. This baseline included the standard NetWare system directories, as well as other directory structures that were common to each machine. There was enough similarity between the five servers tested that only minor modifications were necessary to adapt the standard baseline for server specific directories.

Table 10 - 3 summarises the standard file system baseline that was used for each server. All profiles shown are tree profiles (applying to files in the directory specified, as well as files in sub-trees).

The `SYSTEM`, `LOGIN`, `PUBLIC`, and `PROGRAMS` directory trees were profiled to ensure that executable files were protected by the Read only (Ro) attribute. These profiles also specified that the only allowable executable file owners were administrators

Profile	EXE owners	EXE attributes	Trustee Assignments Permitted
SYS:SYSTEM	Administrators only	At least Ro	None
SYS:LOGIN	Administrators only	At least Ro	None
SYS:MAIL	Anyone	Any	Create (anyone)
SYS:PUBLIC	Administrators only	At least Ro	Read and Filescan (anyone)
* :PROGRAMS	Administrators only	At least Ro	Read and Filescan (anyone)
* :USERS	Anyone		Any except S or A.

Table 10 - 3 File system baseline used during trial project.[5]

(SUPERVISOR or equivalent), or the file server itself. Execute-only, Rename Inhibit, and Delete Inhibit attributes were not used at this site, and were left out of the analysis.

Trustee assignments for the `SYSTEM` and `LOGIN` directories were disallowed, while assignments for the `PUBLIC` and `MAIL` directories were respectively set to the NetWare standards of Read/Filescan, and Create. The project site was conveniently set up with a standard directory naming convention for user and program directories, regardless of the volume on which they appeared. As a result, the two standard tree profiles, `PROGRAMS` and `USERS`, were used for other volumes on the five servers tested.

A variety of profiles were used instead of one blanket profile for the `SYS:` volume. This was because of differing trustee requirements for each directory tree in the analysis (i.e. `SYSTEM` vs `PUBLIC` directory requirements). A further consideration was that NLM and LAN files should only appear in the SYS:SYSTEM directory tree. Separate profiles allowed baselines to be set up to flag instances of these files appearing elsewhere within the file system.

Finally, the file system baseline was configured to check for files with no owners, as well as hidden files and directories belonging to non-administrative users. `SYS:` volume checks were also enabled.

Filesystem results

Results from the file system analysis of the five target servers are summarised in Table 10 - 4 on Page 154. Depending on the server, file system sizes ranged in total size from about 200 Megabytes to over 1.3 Gigabytes. Again, SUPERVISOR and equivalent users are removed from trustee assignment analysis.

[5] The * indicates any server volume containing this directory tree.

Server	A	B	C	D	E
Total size of file system (K)	1330472	1206040	1218488	517852	203024
Number of volumes	2	3	3	2	1
SYS: Check					
Old bindery files	Y	N	N	N	Y
Standard IRM's correct	N	N	N	N	N
Hidden files and directories					
Hidden directories	0	0	0	0	0
Hidden files	7	6	0	0	8
File attributes and owners (standard dirs)					
LAN Files needing Ro	13	16	24	13	0
NLM Files needing Ro	106	53	69	64	13
EXE Files needing Ro	427	224	278	211	138
COM Files needing Ro	8	6	4	3	0
Inappropriate owner/type/location	149	14	17	0	81
Trustee Assignments					
Trustee assignments in SYS:LOGIN.	0	1	1	0	0
Trustee assignments in SYS:SYSTEM	7	9	6	6	1
Write trustee access to SYS:PUBLIC	2	2	0	0	0
Greater than C rights in SYS:MAIL	69	315	0	17	51
S or A access in user directories	83	8	63	1	2
Ownerless files	1676	48	2543	12	0

Table 10 - 4 File system analysis results.

These statistics show that there were a large number of executable files in the SYSTEM, PUBLIC, LOGIN and PROGRAMS directories of these servers that are not protected by the expedient of Ro attributes. In terms of human attackers, this is not a particularly major configurational flaw. However, as mentioned in Chapter 8, the Ro attribute is a last line of defence against file viruses, and its use can help prevent the spread of viral infections. There were no site guidelines regarding the use of file attributes; executables were simply left with the default installation attributes.

Inappropriate file owner and location combination errors accounted for a large number of flagged files on servers A and E. On these systems, non-administrative users had at some stage installed a number of applications in shared program directories. The balance of the files flagged for this problem were attributable to NLM or LAN files residing in unusual (non SYSTEM) directories.

Hidden files and directories did not present a problem for this site. No hidden directories were found, and of the hidden files found, only a small proportion belonged to non-administrative users. These were checked, and turned out to be lock file residue from a popular, but crash-prone, commercial word processing package.

The SYS: volume check revealed that the inherited rights masks for standard directories were not set correctly on any of the servers present. Fortunately, no root trustee assignments were found on those servers. This check additionally revealed the presence of old bindery files on two of the five servers checked.

Analysis of server trustee assignments showed that servers B and C both contained trustee assignments in the LOGIN directory tree. On Server B, this consisted of Read and Filescan rights being granted to the group EVERYONE for the LOGIN directory - not a serious problem, and previously noted by the userbase analysis. However, in the case of server C, EVERYONE was assigned Read, Write, Create and Erase to a LOGIN subdirectory containing executables. This was apparently a commonly accessed part of the file system of that server, and as such would represent an ideal place to plant Trojan horse attacks. This vulnerability was not noted by the userbase assessment, as tagged directories apply to a specific directory - not directory trees.

There were several trustees assigned in the SYSTEM directories of the tested servers. Further checking showed that these assignments were to legitimate objects such as printer servers, queues, and counters. No user trustees were assigned to the SYSTEM directory of any of the systems tested.

A sizeable number of user accounts on servers A and C were assigned Access control rights to home directories (in the USERS directory tree).[6] These rights were mostly assigned to teaching staff user accounts, allowing staff to grant trustee rights to students for specific areas of their home directories. Some students were also assigned Access control rights to their own home directories. Very few Supervisory trustee assignments were located in the USERS trees of the tested systems.

The SYS:MAIL observations in Table 10 - 4 reflect the results from the userbase tagged directory analysis (see Page 150), where Supervisory rights (in the case of E) and Write rights (in the case of A, B, and D) were specified.

[6] These were not included in the tagged directory analysis in the userbase assessment.

Server	A	B	C	D	E
Intrusion Detection settings					
Attempts	10	Inactive	10	Inactive	Inactive
Lockout retention time	1 day		1 day		
Lockout period	15 min		15 min		
Accounting	Enabled	Enabled	Enabled	Enabled	Enabled
SUPERVISOR equivalence	3	3	3	3	2
Workgroup Managers	3	1	1	0	2
Console operators	6	0	0	0	0
Account defaults					
Expiry date	None	None	None	None	None
Minimum password length	None	None	None	None	None
Maximum concurrent logins	2	2	2	Unlimited	Unlimited

Table 10 - 5 Summary of the general server setup results.

A final observation about the file system security of the tested systems was that there were significant quantities of ownerless files present on servers A and C. These reflected the number of accounts that had been created and destroyed on these servers over the lifetime of their respective file systems.

GENERAL ISSUES

A summary of other more general security information from NetAudit server reports is given in Table 10 - 5. As can be seen from this table, there were inconsistencies in the use of the intrusion detection lockout mechanism, as well as the use of account setup defaults.

Of the servers tested, only A and C enabled the lockout feature; the other three servers did not use this mechanism. This is understandable for server E, given its experimental nature, but is unusual for servers B and D. No reason was given by the administrators of these systems as to why lockout was disabled.

That the account setup defaults were configured in this fashion was also unusual. Usually, accounts are generated automatically, with the correct password and concurrent login settings specified by the generating program. However, accounts that are setup manually (in SYSCON) will still use these defaults, which need to be overridden by the administrator creating the account to ensure conformance with site standards.

On a more positive note, Table 10 - 5 shows that the accounting feature was enabled on all servers. In addition, this table shows that there were an acceptably low number of SUPERVISOR equivalent and workgroup manager accounts active. Server A was the only server that had users with console privileges.

SUMMARISING TRIAL PROJECT SITE SECURITY

The analysis of the test servers revealed little sign of past attacks or misuse. However, as discussed previously, a number of areas of configurational vulnerability within the userbase and file systems of these servers were uncovered.

There was evidence of a lack of consistency in the use of NetWare security mechanisms between servers, particularly in the administration of user accounts. For example, some accounts had minimum password lengths specified, while some did not. Several non-guest user accounts on a variety of servers lacked passwords, or login scripts. On some servers, users were granted access controls to their own `MAIL` or `USER` directories, while in other cases they were assigned only minimal rights to these directories.

File system controls were also not used to their full advantage; file attributes were not uniformly used, and trustee assignments for shared directories were sometimes insecure. Anomalies in the use of the intrusion detection lockout features, as well as account defaults, were also noted.

A discussion with the administrators at the test site established that no formal site policy existed within the organisation outlining security requirements for servers connected to the organisational LAN. The results of such a lack of guidelines are reflected in this trial project, where many inconsistencies have been noted between systems. These results reinforce the notion that a formal or semi-formal site policy outlining acceptable use of system security mechanisms will almost certainly reduce the number of configurational vulnerabilities that are present.

It is important to note that NetAudit baseline developed for this site was not optimal. For example, password and account expiry mechanisms, node and time restrictions, and detailed trustee assignment data confidentiality checks, were not used. However, the baseline did serve to both detect vulnerability where it existed, as well as compare all five servers to the same standard.

10.2 NETAUDIT EVALUATION

As mentioned at the beginning of this chapter, the objectives of the trial project were not confined to simply testing the security of the selected site. Two other objectives of the project were to evaluate the value of baseline checking (discussed below), and to comment on the suitability and effectiveness of the NetAudit tool (see Page 159).

BASELINE APPROACH EFFECTIVENESS

On the whole, the baseline approach appeared to be quite successful. While the majority of objects tested in the sample project conformed to the minimum standards set by the site baseline, those that did not were detected, and were able to be further examined by the author for vulnerability. Often, it was found that objects not conforming to baseline standards in one area were also vulnerable in other areas. As discussed in the previous section, a number of outright vulnerabilities were also identified. It is important to note that further study of this technique within other types of site is necessary before any concrete conclusions can be drawn about the effectiveness of this approach. Some general observations about NetAudit's baselines are discussed below.

Overall, the most valuable aspect of the baseline approach is that objects not conforming to site security standards are detected. As mentioned elsewhere in this thesis, such standards may vary quite significantly between different types of organisation, or even within different departments of a single organisation. Configurable baselines, such as those supported by NetAudit, can accommodate these varying needs.

It is, however, possible to specify a baseline that is inherently insecure. As a simple example, a NetAudit baseline may be configured to not check for NULL passwords. When used, such a baseline will not report any problems for those accounts lacking passwords. The importance of this point is that while NetAudit is puportedly providing assurance that the system conforms to a given set of standards, those standards are themselves insecure. Indeed, there may be a case for the creation of an analysis tool that checks the configuration of configurational audit tools.

By ensuring inter-object and inter-server consistency and standards conformance, the auditor can be more certain that there are less opportunities for vulnerability to escape unnoticed. Systems conforming to a straightforward, yet secure, baseline setup are less likely to suffer from complexity in the use of security controls. In the case of NetWare, for example, the more trustee assignments and equivalencies there are, the greater the chances are that unforseen interaction within the file system will become a problem. By reducing

the number of individual object trustee assignments, and reducing the number of inter-user equivalencies, the security setup of the target system becomes easier to understand and administer.

Baselines go hand in hand with organisational security policy. A clearly defined policy on the use of technical security mechanisms can be used as a template from which baseline configurations are extracted. At sites lacking such policies, such as that analysed in the trial project, inconsistencies in the use of these mechanisms are likely.

NETAUDIT EFFECTIVENESS

There were a number of problems encountered while using NetAudit. As it stands, reports produced by NetAudit from the trial analysis proved to be inadequate for post processing. This was obvious from the amount of off-line text processing that was required to massage these reports into meaningful statistics for the analysis given in the previous section. Several minor and not so minor bugs in the implementation of the package were also located and fixed during the analysis project.

The simple ranking system for user account vulnerability reports proved to be very useful. By ordering vulnerable user accounts by the number and seriousness of the non-conformities or vulnerabilities detected, it was possible to get a reasonable idea of the more seriously vulnerable accounts on each server.

This mechanism could be extended to simulate multiple vulnerability checking. This could be achieved by adding a large figure to the total account "score" when particularly bad combinations of vulnerability for a single user are detected.

Extensions to NetAudit to check the contents of login scripts and batch files in system directories are necessary. While not strictly a conformance test, such checks will certainly help detect some of the more subtle vulnerabilities that may occur when trusted executables, scripts or batch files reference unprotected publicly accessible components. As mentioned in chapter 9, these will be added to a later version of the NetAudit package.

The NetAudit user interface needs to be redesigned to support automated and unattended scans of multiple servers. The browsers are useful for interactive testing, but the lack of unattended checking facilities proved to be quite a burden where there were several servers to check.

More complete checks are required for file objects. For instance, as mentioned in Chapter nine, NetAudit assesses file type based on the extension of the file. There are obvious flaws in this approach; a better method would be to attempt to discern the contents of the file based on its contents. Most executable programs have a fairly well-defined structure, as do graphics, wordprocessing and text files.

Extending NetAudit to include checks of client workstations may be possible. By running an automatic agent as part of the system login script, every client could be subjected to a "mini-scan" as it logged in. Unfortunately, NetWare clients can easily bypass system and user login scripts during the login process - work needs to be carried out to ensure that such workstation checks are bypassed by clients as little as possible.

CHAPTER 11
SUMMARY AND CONCLUSIONS

For whatever reason, vulnerability is a common feature of many computer systems. Half the battle in the security process, it seems, is recognising that vulnerability can and does appear, even in systems with outwardly impressive controls. By assuming that vulnerability exists, one is in a better position to approach the problem of locating such flaws with an open mind.

Although vulnerability may occur in any of the security environments discussed in Chapter two, this thesis has focussed on the configurational weaknesses that often plague technical controls. Such flaws occur easily, and yet can lurk undetected within systems for long periods of time before being discovered and exploited (or fixed, depending upon who discovers the flaw). From a risk analysis perspective, these weaknesses are extremely cost effective to safeguard against, simply because the costs of reconfiguring security controls, or installing vendor patches, are so low.

Chapter three of this thesis presented a general overview of technical vulnerability, and showed that technical security weaknesses can occur because of a variety of both general and system life-cycle factors. Chapters four, five, and six discussed the techniques and philosophies of some existing tools designed to detect the presence of configurational vulnerabilities.

Chapter seven introduced NetAudit, a tool prototype designed to address configurational vulnerability within Novell NetWare 3.1x systems. As established in this chapter, NetWare exhibits many of the same factors that lead to configurational weaknesses in other computing environments. Chapter eight provided an analysis of how configurational vulnerability might manifest itself within the NetWare environment, while chapter nine described how those results assisted the design of NetAudit.

Chapter ten presented a trial project, in which NetAudit was used to assess the security of a small number of NetWare servers at a production site. A number of security anomalies and outright vulnerabilities were discovered within these systems, and major inconsistencies in how these servers applied NetWare security mechanisms were noted. These inconsistencies illustrate the importance of organisational policies regulating the use of technical security controls. The lack of such guidelines can mean that ad-hoc choices may be made about the use of these controls, resulting in potential mayhem at some stage in the future of the affected system.

The evaluation of the NetAudit approach given at the end of chapter ten suggests that conformance and baseline checking are valid techniques for detecting configurational vulnerabilities. This is only a tentative conclusion; a wider range of sample sites are required to confirm these results.

Nevertheless, common sense tells us that if the controls of a system conform to secure and uniform standards, then the chances of vulnerability are lessened considerably. Of vital importance, then, is deciding what constitutes a sufficiently secure standard for the purposes of a given target system. This is essentially a policy question, and NetAudit is a tool that uses such guidelines to confirm that system controls adhere to those policy standards.

Some caution is also required. One should remember that configurational audit tools merely give an opinion about the presence of *known* vulnerabilities. As with any remedial approach to assessing computer vulnerability, NetAudit provides no guarantee that assessed systems are totally free of technical flaws. Configurational audit and conformance tools can only ever assess the presence of potential weaknesses that they are aware of, and new flaws in technical safeguards are discovered almost daily (although not necessarily in the NetWare environment).

In summary, it is safe to say that the best way to deal with complex computer security mechanisms is to discipline the use of those controls, embracing simplicity and uniformity wherever possible. Configurational audit tools, especially those incorporating conformance detection components, can help achieve this.

FUTURE WORK

As discussed in chapter ten, the NetAudit prototype does suffer from some implementation weaknesses. Some of these have been remedied, while others await further effort on the

part of the author. In addition, a number of possible enhancements were also suggested in *NetAudit effectiveness*, on Page 159. Two other issues, the application of NetAudit techniques to other LAN server environments, and the use of other configurational audit techniques, are discussed below.

Other platforms

In its current form, NetAudit is specific to NetWare 3.1x. Due to the differences between NetWare's security mechanisms, and those of platforms such as Microsoft's Windows NT or IBM's LAN Server, it is currently impractical to develop a tool that competently addresses all three. Nevertheless, these platforms face many of the same configurational security problems of NetWare systems, and so developing suitable configurational audit tools for each would be a worthwhile exercise.

The NetWare 4.1 operating system, a considerably enhanced version of NetWare released by Novell in 1993, has many features in common with earlier versions of NetWare. File system controls are in some cases identical, and trustee assignments are still used. The biggest difference between the two is that NetWare 4.1 replaces the bindery database with a more scalable mechanism called Directory Services. This mechanism allows enterprise-wide management of LAN resources, and is considerably more complex than the simple bindery mechanism described in this thesis.

Due to the similarities between versions 3.1x and 4.1, much of NetAudit's analysis can be directly applied to the NetWare 4.1 environments. Extensions are required to address Directory Services configurational issues, but these would simply require the addition of another module to NetAudit.[1] Given the complexities of Directory Services, the need for configurational audit tools for NetWare 4.1 systems is just as pressing as it was for earlier versions of NetWare. Notwithstanding this, an analysis of NetWare 4.1's controls, similar to that described in this thesis, will still need to be carried out to assess potential sources of vulnerability.

Other techniques

Currently, artificial intelligence techniques are not widely applied to configurational audit tools. The one exception discussed in this thesis was the Kuang tool, a basic system for

[1] NetWare 4.1 does offer a bindery emulation mode that allows 3.1x clients to use bindery services on a Directory Services server. Unfortunately, the emulation is not sufficient to allow the current version of NetAudit to accurately assess server security. In addition, there are many new NetWare services that are not dealt with by this bindery emulation mode.

assessing individual account vulnerability (see the discussion of COPS, in Chapter 6). This tool showed that there is a place for rule-based expert systems in configurational audit.

More work is required to apply such techniques to the problem of detecting, in particular, multiple vulnerabilities. The relationships between individual vulnerabilities can sometimes be subtle; an expert system can encapsulate multiple vulnerability knowledge, and recognise situations that may otherwise escape attention during a conventional configurational audit. Work is also required to make such expert systems easier for auditors to access and use.

Graphical visualisation of security controls could be an important factor in future tool designs. This visualisation could allow the auditor to build a more accurate mental model of system security configuration. Such tools may assist the auditor to quickly understand how security is enforced on the target system, and where vulnerabilities may be present. The NetAudit tool, for example, uses a limited graphical display to provide the auditor with a view of the file system from the point of view of a user. This concept could be expanded on to provide a "tour" of security mechanisms from the point of view of a particular security object. Currently, very few configurational audit tools employ graphical user interface techniques.

APPENDICES

APPENDIX A
IDENTIFYING TECHNICAL VULNERABILITY

This appendix presents a brief overview of some of the methodologies, techniques and sources of information that can be used to assess the presence of technical vulnerabilities within existing production systems.

A.1 INTRODUCTION

According to an early paper by Nuemann [50], there are essentially two approaches used to deal with technical vulnerabilities within a system: *preventative*, and *remedial*. Preventative approaches attempt to reduce system vulnerabilities by avoiding (or identifying) them at the design and implementation stages. Remedial approaches are applied to existing systems that are in use - but which may contain vulnerabilities. Although Nuemann's paper was written in 1978, both approaches are, by and large, still valid today.

The TCSEC [20] and ITSEC [31] evaluation criteria both note that systems must be *designed* in a secure fashion in order to receive a high assurance security classification. A system that has been formally specified and verified, for instance, is more likely to have been subjected to exhaustive tests validating its security model, and the mechanisms implementing that model. Conversely, systems that have been designed without the benefit of a formal pedigree lack the necessary design simplicity required to exhaustively test their security. Not surprisingly, these systems tend to receive lower classifications. [20, section 6.4]

Remedial approaches are applicable where security controls have been *added* to a system, rather than integrated as a fundamental design objective. Most modern commercial operating systems (i.e. standard Unix, Novell NetWare, MS-DOS, standard DEC VMS, etc.) fall into this category. Remedial approaches attempt to uncover system vulnerabilities by testing system controls or analysing system source code.

The best results obtainable from remedial approaches are that *some* existing vulnerabilities may be uncovered. After the first few vulnerabilities are discovered, successive flaws will become increasingly difficult to detect. An exhaustive examination of the security mechanisms is unlikely, due to a lack of a formal "roadmap" for those mechanisms. To make matters worse, safeguards implemented to remedy discovered flaws may themselves contain further vulnerabilities, complicating the entire process [50].

Nevertheless, remedial approaches are still useful. Even if such analysis fails to identify all system vulnerabilities, the net result is (hopefully) that the system will be more secure than it was prior to the analysis.

A.2 THE FLAW HYPOTHESIS METHODOLOGY

A classic approach to the problem of locating operating system flaws is embodied in the Flaw Hypothesis Methodology (FHM). First formalised by Richard Linde in 1975, the FHM reportedly enjoyed a penetration success rate of up to 65% on systems tested [38]. Flaw Hypothesis Methodologies are still commonly used to evaluate security (i.e. see [33, p.263]). An updated approach to FHM is given in Pfleeger *et al.* [53].

Flaw hypothesis methods can be employed at almost any level of abstraction. For example, a penetration analyst might hypothesise that an access control flaw exists that allows an unauthenticated user access to confidential information. On a different level, an analyst might hypothesise that a certain system call, when passed a given parameter, will produce a kernel panic. While ostensibly testing for flaws introduced at the implementation or production stages of a system's life, vulnerabilities discovered during a flaw hypothesis analysis may also help reveal underlying weaknesses in system design.

A basic assumption of the flaw hypothesis methodology is that certain types of generic vulnerabilities will exist on most systems. This assumption has proved reasonably successful, as indicated by the reported success rate in [38]. In the case of operating systems, the functional categories in which vulnerabilities will most likely occur are:[1]

- I/O Control.
- Program and data sharing.
- Access control.
- Installation management and operational control.

[1] A comprehensive list of common flaws is given in [38, Appendix A].

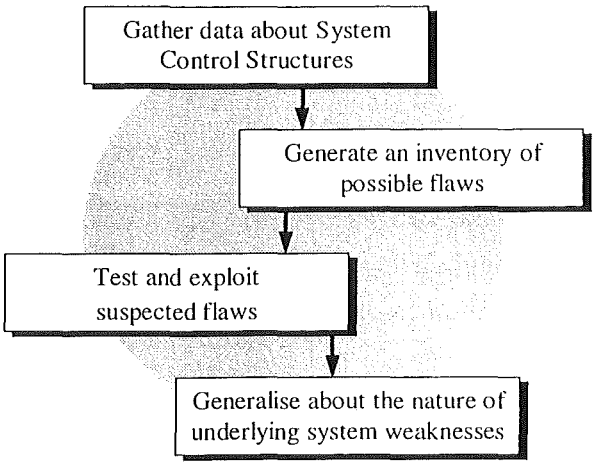


Figure A - 1 The Flaw Hypothesis Methodology.

- Auditing and surveillance.

Using knowledge of the system being evaluated, the penetration analyst hypothesises the existence of likely flaws, and then devises and executes a set of tests to support or refute each hypothesis. Hypothesises proved correct may lead to generalisations about underlying system weaknesses. Flaw Hypothesis Methodology steps are summarised in Figure A - 1, and are detailed below:

1. *Gaining knowledge of system control structures*

   The first FHM stage involves gaining an in-depth knowledge of the system. This may include reading design manuals, implementation manuals, programming library documentation, system documentation, operations documentation and procedural documentation.

   The objective of this step is to build a comprehensive picture of the objects that are protected by the system, and the controls that protect them. Some objects may exist in both classes. For example, Unix `/etc/passwd` files exist as both a protected object, as well as a control object that protects other system objects (in this case, user accounts).

   The advantages of identifying important control objects within the system are twofold: First, during the flaw hypothesis generation step (see below) the penetration analyst can apply generic templates of "good" and "bad" system control object arrangements, in an effort to ascertain the absence or misplacement of safeguards. Second, identified control objects can be subjected to more rigorous testing.

   The level of abstraction at which penetration tests are carried out will dictate the type of information gathered at this stage. For example, if the analyst is interested in testing

flaws in an operating system's implementation, then data concerning internal system structures (i.e. process tables, kernel data structures, operating system calls) will be the most useful. Alternatively, the analyst may be interested in testing the effectiveness of system controls at the level of an untrusted user, and hence may collect information about how general users interact with these controls.

Gathering information about control structures may or may not involve an examination of the system source code. In some cases, source for the system under investigation may not be available, and it will be necessary for the penetration analyst to hypothesise about the nature of internal system structures by examining published system interfaces. If source code is available, tools such as source analysers, cross-referencers, and text searching programs may be useful to help search and organise code.

2. *Generating an inventory of suspected flaws*

   At some stage the penetration analyst will have gathered enough information about system control structures to begin generating a list of suspected flaws. Usually, comparing control objects to generic (but common) "bad" templates will yield a good number of hypotheses to work on.

   As noted by Muffett [41], the genealogy of the system may present other useful hypotheses. For example, the system currently under evaluation may be derived from an earlier system that was known to contain a flaw. In this case, a good hypothesis may be that the same flaw has made it unscathed into the system currently being evaluated.

3. *Confirming Hypotheses*

   There are two ways that can be used to confirm or refute hypotheses generated in the previous step. The first involves "gerdanken", or thought experiments. In these experiments, the analyst works from the system documentation, source code, or other information and proves on paper that the flaw hypothesis is valid.

   The second method is a live test of the flaw hypothesis. This may be required if gerdanken testing proves inconclusive. For example, the flaw may involve asynchronous code problems that may only appear under certain system conditions. In cases such as these, there may be no choice but to go ahead and write a program or test suite to exercise the suspected flaw.

4. *Making generalisations about underlying system weaknesses*

   The last step in the Flaw Hypothesis Methodology involves reviewing successful penetrations and identifications of flaws, and generalising about the strengths and

weaknesses of the system's security mechanisms. From these reviews, further flaws may be hypothesised and subsequently tested.

DRAWBACKS

While the Flaw Hypothesis Methodology is a useful strategy for finding vulnerabilities within systems, one major difficulty with the approach exists. Flaws hypotheses are generated by hand on an *ad-hoc* basis, and consequently systematic and complete exercising of the security mechanisms is not attainable using this methodology.

The number of hypotheses generated are fundamentally limited by the analyst's level of knowledge of the system under evaluation, and by the quality of any "educated guesses" the analyst may make.

In some cases, though, a more thorough method of generating hypotheses is required. Where source code is available, source code analysis may be used to generate and test an exhaustive list of hypothesis. This is sometimes known as "white box" testing. Examples and experiences of a source code analysis tool are given in Gupta *et al.* [30].

Alternatively, where source code or detailed system information is not available, "black-box" testing may be used. Black box testing assesses the relationships between system inputs and outputs. This analysis may reveal details of the internal system design, from which flaw hypotheses can be generated. Inconsistencies or vulnerabilities may also be directly revealed by this approach (Chorley [11, p.45]). Black box testing is generally a last ditch technique.

As pointed out by [11], the results gleaned from black-box tests are at best speculative. In complex systems, black box testing may uncover some flaws, but is just as likely to miss many more. Nevertheless, as with most remedial approaches, black-box testing still retains some value simply because it may be the only applicable methodology.

A.3 TIGER TEAMS

Tiger teams are a useful, if somewhat controversial, method of assessing the security of computer systems. Tiger teams proactively attempt to penetrate the system in order to assess the effectiveness of security safeguards, and to detect vulnerabilities that may have been missed in regular risk analysis and audit exercises.

Generally, tiger team projects may be initiated by one of three groups: [28]

- *Upper management*

  Upper management may be nervous about the security of their computer system, and may want an independent analysis performed to provide additional assurance.

- *Middle management or system auditors*

  Occasionally, middle managers may commission a tiger team project in an effort to collect concrete information about system vulnerabilities in order to convince upper management to pay more attention to security issues. Alternatively, company auditors may commission a project as a way of assessing the vulnerability of company information systems, or to add weight to some other audit findings (the "I told you so" approach [29]).

- *Security officers*

  The third group who may commission a tiger team project are those responsible for the security of the target system. Often, this may be to add a "real world" dimension to risk analysis exercises, or to confirm their opinion that the security of the system is adequate (the "Take your best shot" approach [29]).

TIGER TEAM APPROACHES

The general approach adopted by tiger teams varies depending on the goal of the assessment. The goal may be to successfully gain access to some object (i.e. a specific file), or to some privilege (i.e. superuser privileges). A time limit of perhaps a few weeks is imposed upon the project.

As the following excerpt (from [74, p.46]) illustrates, tiger teams tend to assess the *overall* level of system security, not just the technical security issues:

> *... [the tiger team] was hired to prove that the security systems were weak at a U.S. government defence contractor. [The] tiger team electronically broke into the network in less than 10 minutes using a dial-up modem and left messages on the network to prove they were there. Through windows, the team also took photos of classified documents displayed on computer terminals. And a man dressed in a white uniform was sent in carrying a water bottle and was allowed to enter the building without a security check. When inside, he infected computers with a fake virus.*

In the above example, the tiger team considered vulnerabilities in network access controls, physical access controls and protection, and access controls to workstations.

Initial viewpoint

Tiger teams can assess security from two points of view: *outsiders* or *insiders*. When assessing security from an outsider's viewpoint, members of the team are told nothing about the system. From this limited vantage point, they then have to learn enough about the system to break into it. This viewpoint does have disadvantages, as pointed out by Goldis [28, p.5]:

> *Only false comfort would be gained by having the tiger team conclude after two weeks of trying that they did not have the right modem or could not guess the right telephone number.*

From an insider viewpoint, the tiger team is given the access privileges and system information that is afforded a "normal" system user - for instance an employee. This is probably a much more effective method than the outsider approach, given the rather larger risks presented by inside threats. The emphasis therefore switches to determining that the internal safeguards protecting system assets are in place, and are functioning correctly.

As noted by [29], several intermediate vantage points exist. For instance, the attacker could be an outsider who has a little information about the company that has been learned from ex-employees; or the attacker could already be a privileged user, such as the super-user. Assessing security from these various viewpoints helps to build a better picture of overall system weaknesses.

Covert or overt

The decision over whether to operate the tiger team as a covert or overt operation is an influential one. If the operation is operated covertly, members of the system staff may become upset once it is revealed to them that people have been assessing "their" system behind their backs. On the other hand, the tiger team is likely to get a more representative sample of security practices if they are allowed to assess them secretly. If the operation is run overtly, system staff may increase their attention to overall security, and be more alert for potential vulnerabilities during the project. This is not necessarily a bad thing.

Scope

The scope of the tiger team project may take into account just the computer hardware and software, or it can take into account the entire security environment. For instance, during one tiger test exercise of the Multics operating system, penetration analysts sent a bogus system update tape containing a Trojan horse to Multics site. Due to lapses in procedural

security, the tape was duly loaded, and the team were subsequently able to access the operating system (Cooper [13, P.222]).

Methods

Tiger teams are able to use any of the techniques that a potential attacker might use. For instance, they may adopt a flaw hypothesis methodology and attack the system interface directly; they may use black box testing on system applications or utilities; they may employ network monitoring tools to capture unencrypted passwords; or they may employ subterfuges such as calling the system help desk and requesting a new password for a targeted user account.

Goldis [28] mentions one other specific attack technique that may be employed by tiger teams: *probing*. Probing is where the analyst uses the standard privileges of a user, and browses the system for questionable access.

DRAWBACKS

There are several drawbacks to the tiger team approach. First, as noted in [74], tiger teams by their nature are haphazard approaches, and should not be viewed as a replacement for conventional security assurance tests, such as audits and risk analysis.

The tiger team may not find any significant flaws in the security of the system within the allotted timeframe. This in itself does not prove that the system is secure, but that simply either the team did not have the level of skills required to penetrate security within the timeframe, or that any vulnerabilities present are well hidden.

Tiger teams have received some negative reaction within the security community, especially when "reformed" crackers began offering their services as security experts to the wider business security[2]. Reputable providers of security services, such as Coopers & Lybrands and Price Waterhouse, avoid using ex-crackers because of the associated risks involved with cracker personalities[74].

Tiger teams must also be careful to avoid becoming so overzealous in their efforts that they interrupt normal system processing. For instance, proving that the system is prone to denial of service attacks from user created programs by creating and running a program that consumes all system resources is not a recommended strategy.

---

[2] Such as members of the infamous "Legion of Doom" hacking team.

A.4 OTHER VULNERABILITY DISCOVERY METHODS

Apart the methods described in the previous sections, vulnerabilities may be discovered in production systems using a variety of less systematic means. This section discusses methods whereby an administrator can discover vulnerabilities within their own systems. These methods include:

- *Configurational Audits*

  Configurational audits are discussed in this thesis. In a configurational audit, the underlying strengths and weaknesses of system control mechanisms are not considered; rather, these audits attempt to assess whether system controls are configured to provide an acceptable level of security. Configurational vulnerabilities are one of the most common vulnerabilities exploited by attackers, and identifying them within a system is an important aspect of system security.

- *Monitoring*

  Monitoring the activity of active computer attackers is a technique that can yield valuable information about vulnerabilities that are exploited during attacks, as well as the attack methodologies used. Bellovin [4] and Cheswick [10] both describe monitoring strategies that were employed to track the activities of attackers attempting to gain access to AT&T Bell's internal computer network. By observing attacker activity, both from system logs, and from within a specifically constructed fake environment, these administrators were able to build a reasonable catalogue of a number of standard, as well as more sophisticated, Unix attack methods.

  Basic system monitoring is another method of gathering such information. Unfortunately, as noted by [4], more sophisticated auditing mechanisms and data reduction tools are required to identify attack attempts and suspicious behaviour. Large systems may typically generate between 1 and 20 Megabytes of audit data per day, an amount impossible to analyse without the aid of tools.

  Intrusion detection systems[3] are an emerging technology that may provide an answer to the problem of identifying attack behaviour from volumous amounts of audit data. By automatically analysing audit information, intrusion detection systems attempt to identify system attacks as they occur. Intrusion detection systems may also detect

[3] For a comprehensive overview of Intrusion Detection, see Lunt[40] or Watson[79].

usage anomalies, which may indicate that an attack is occurring using a vulnerability not known to system users or administrators.

- *Trawling*

  A good approach for those interested in learning about system vulnerabilities is to read those documents and books from which crackers learn their craft. Many sources of information exist detailing specific system vulnerabilities, and information about the people who exploit them. These sources include underground electronic journals (such as Phrack, 2600 etc.), and security related net-news groups (i.e. alt.2600, comp. security.misc, comp.security.unix, and comp.risks). There are also many repositories of security related information around the Internet, such as `spy.org`, `cert.org`, and `csrc.ncsl.nist.gov`.

- *Vulnerability watchdog organisations.*

  Organisations such as CERT (Computer Emergency Response Team) and CIAC (Computer Incident Advisory Capability) periodically publish warnings about newly discovered vulnerabilities in various systems (see Scherlis *et al.* [68] and Schultz *et al.* [69]). These security advisories are available from the ftp sites mentioned above, or from various mailing lists.

  CIAC and CERT advisories are typically "terse" explanations, that purposefully do not give much information on the nature of the vulnerability discussed. The 8LGM list server, `fileserv@bagpuss.demon.co.uk`., publishes more detailed advisories of common security flaws. Access to these advisories is subject to more controls than the public distribution of CERT or CIAC advisories.

APPENDIX B - A VULNERABILITY ANALYSIS OF NOVELL NETWARE VERSION 3.11

This appendix presents an analysis of Novell NetWare Security based upon a Local Area Network Security architecture proposed by Lisa Carnahan of the U.S. National Institute of Standards and Technology. By applying this risk analysis framework to NetWare 3.11 and 3.12 LAN security, a number of potential areas of vulnerability in NetWare systems are identified and discussed.[1]

B.1 THE LAN SECURITY ARCHITECTURE

Local area networks pose a unique problem to information security. On one hand, it is difficult to deny the useful nature of the LAN architecture. By distributing computing functionality around an organisation, the network provides a flexible and easily expandable computing environment. Network users are able to share resources such as storage space, applications, database services, communications services and access to other host systems. The network also allows users to exchange messages and form work-groups of people working on related projects.

On the other hand, a LAN architecture discourages centralised control. Security policies and mechanisms that work well on a centrally managed system may not be so effective in a networked environment. Networks typically allow a greater number of paths to information, and not all of these paths may be secure ones. Improperly managed networks are more likely to suffer exposure to threats both internal and external to the organisation. Poor controls or a lack of understanding of network security issues can lead to a computing environment that is vulnerable to a myriad of threats.

[1] This appendix was originally a technical report prepared by the author for the New Zealand GCSB.

Analysis Category	Description
Unauthorised LAN Access.	The threat of unauthorised parties gaining access to the LAN.
Unauthorised Access to LAN Resources.	The threat of unauthorised parties gaining access to LAN resources.
Compromise of LAN Data.	The threat of unauthorised parties gaining access to LAN data.
Unauthorised Modification of LAN Data and Software.	The threat that unauthorised changes are made to LAN data and software.
Compromise of LAN traffic.	The threat that LAN traffic is intercepted as it travels across the LAN medium and is used to obtain unauthorised access to information.
Modification to LAN traffic.	The threat that Data transmitted over the LAN is modified.
Spoofing of LAN traffic.	The threat that LAN traffic is intercepted, by masquerading as either the legitimate sender of the data, or as the legitimate receiver of the data.
Disruption of LAN functionality.	The threat that LAN services are disrupted.

Table B - 1 Categories of LAN vulnerability Analysis.

Lisa Carnahan's Local Area Network Security Architecture ([7], [9]) provides a set of guidelines and procedures for analysing risk in LAN systems. As part of this risk analysis, the security analyst is required to identify individual threats to the LAN, the vulnerabilities in the LAN that may be exploited by the threat in question, and the safeguards that are in place or available to reduce the risk of those vulnerabilities.

This report starts with a discussion of the general architecture of the NetWare LAN operating system, followed by an examination of the threat areas discussed in [7]. For each threat area, the relevant NetWare technical safeguards are identified; potential vulnerabilities that may be exploited by an attacker in order to realise threats are also identified. The threat areas examined (based on [7]) are summarised in Table B - 1 below. This report relates only to NetWare versions 3.11 and 3.12, and assumes that no Novell or third party security enhancements have been added. Certain problems with NetWare security that are outlined in this report may or may not have corrective patches available from Novell.

In this appendix, items that are of importance to the overall security status of a NetWare network will appear in this font.

B.2 NETWARE SYSTEM ARCHITECTURE

The NetWare 3.11 and 3.12 operating system runs on dedicated file servers attached to the network. Access to the operating system is granted to network users who satisfy authentication requirements. Programs running on the NetWare file servers, called NLM's,

provide core NetWare functionality, as well as additional network services. These processes are cooperatively multi-tasked, and run at the highest level of privileges that NetWare supports.

Two NLM issues are important. First, because these programs run at elevated privileges, they represent an easy way for an attacker to subvert the security mechanisms. Second, because NLM's are multi-tasked cooperatively, a single buggy NLM that crashes without relinquishing control of the CPU will cause the entire file server to cease functioning.

THE NETWARE BINDERY

NetWare keeps track of network information in a structure known as the bindery. The bindery is a flat-file database which contains information about the network operating environment, network users, network user security settings, restrictions and system accounting data. The bindery is composed of objects, properties and property data sets.[2]

Objects

An object can be a user, a group of users, a file server, print server or any other logical or physical entity on the network that has been given a name. Each bindery object is assigned a unique 8 byte bindery ID by the file server at creation time. By default, bindery IDs are not coordinated between servers on multiple server networks.

Properties

Each bindery object has a number of characteristics, or properties. These can include such data sets as passwords, account restrictions, account balances, groups that the object belongs to or a list of authorised clients (in the case of a network service). For instance, user accounts have a number of properties associated with them, such as a PASSWORD property, a SECURITY_EQUALS property and so on.

Property data sets

Property Data Sets are the values that are assigned to bindery object properties. Some bindery properties may have only one value associated with them, as in the case of a PASSWORD property; others, such as a SECURITY_EQUALS property may have a number of values.

---

[2] Objects, properties and property data sets are stored in the the system files NET$OBJ.SYS, NET$PROP.SYS and NET$VAL.SYS respectively. Together, these three files make up the bindery database.

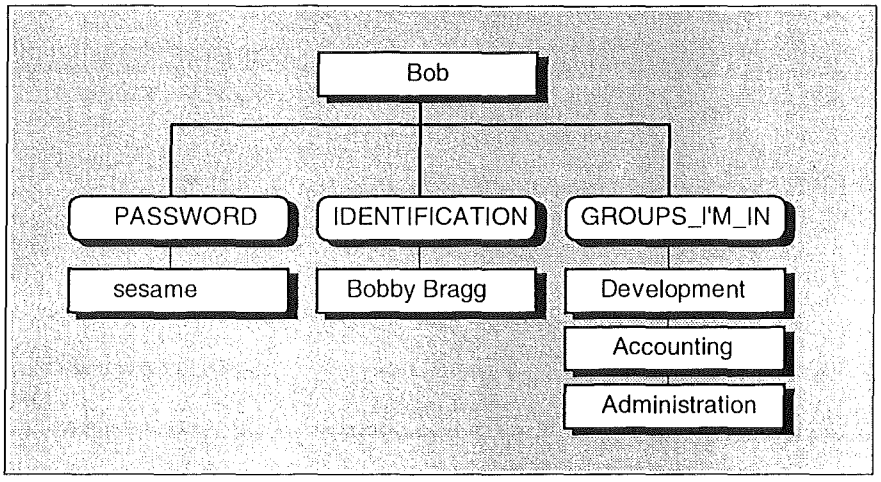


Figure B - 1 Bindery database representation of user object BOB (not all user properties are shown).

As shown in Figure B - 1, the bindery user object BOB has three properties: PASSWORD, IDENTIFICATION and GROUPS_I'M_IN. Associated with the property PASSWORD is the single value *sesame* (stored in encrypted form in the bindery), while the IDENTIFICATION property contains a full name for Bob. The GROUPS_I'M_IN property shows that Bob is a member of the *Development, Accounting* and *Administration* groups.

BINDERY OBJECT SECURITY

Bindery security is enforced by assigning security levels to bindery objects and properties. The security level dictates who may read or write a bindery object (using the Object Security Level or OSL) or an associated bindery property (the Property Security Level or PSL) for which the security level is effective. The OSL and PSL are stored within the bindery as a one byte value directly associated with an object or property.

This value is further divided into two nibble sized fields (four bits each); the low order nibble controls read access, while the high order nibble controls write access. These fields can assume any one of five values, as described in Table B - 2.

Binary	Access	Description
0000	ANYONE	Access is allowed to all clients that have attached to the file server, but have not logged in (ie have loaded the NetWare shell at a workstation, but have not supplied a user name and password yet).
0001	LOGGED	Access is allowed only to clients who have logged into the file server.
0010	OBJECT	Access is allowed only to clients who have logged into the file server with the same object name, type and password.
0011	SUPERVISOR	Access is allowed only to those clients who have logged into the file server as SUPERVISOR or as a supervisor equivalent user.
0100	NETWARE	Access by the NetWare operating system only is allowed.

Table B - 2 Bindery access levels for objects and properties.

A client's bindery security level (BSL) may in practice assume only three of these levels: ANYONE, LOGGED or SUPERVISOR. This level is initially set to ANYONE when the client attaches to the network, and changes to LOGGED or SUPERVISOR once the client has successfully logged into the file server as either a normal user or a supervisor equivalent user. The system thereafter differentiates only between the LOGGED and SUPERVISOR levels.

Access to bindery objects is possible only via NetWare's bindery services APIs. These provide client workstations with regulated access to bindery information, maintaining the security of the bindery database. There are four main interactions that can occur between a client and bindery services. These are shown in Table B - 3.

B.3 UNAUTHORISED LAN ACCESS

The threat of unauthorised LAN access occurs when an unauthorised party gains access to services of the LAN. Because resources such as disk space, printers, scanners, modems,

Interaction	Checks made.
Read access to a bindery object.	The low order nibble of the OSL associated with the bindery object is checked. It must be at least LOGGED for a client to be able to see it (assuming that the client is currently logged in).
Write access to a bindery object.	The high order nibble of the OSL associated with the bindery object is checked. Here, the value must at least be LOGGED before anyone can write to it. If the value of the OSL is SUPERVISOR, then the user must have a BSL of SUPERVISOR in order to write to the object.
Read access to a property.	The low order nibble of the PSL is checked, and interpreted in the same way as above.
Write access to a property.	The high order nibble of the PSL is checked, and processed as per objects. In this context, write access allows a value to be added to the item/set property. Interpretation of the values is accomplished in the same manner as previously mentioned.

Table B - 3 Summary of bindery security level interactions (summarised from Lamb [39]).

faxes and communications gateways are shared, some form of control must be implemented in order to regulate access to those devices and services, and to ensure that a basic level of resource accountability exists.

The most common form of attack employed by an attack agent to realise this threat is to subvert or bypass the identification and authentication mechanisms. This approach can take many forms. In NetWare an attacker can take advantage of misconfigured or non-existent controls; can steal passwords directly from another user using fake login programs, social engineering, or "shoulder-surfing"; or in some circumstances can use a packet sniffer to scan the network medium for password bearing packets.

The consequences of this threat occurring can be related to the value of the data and software stored on the LAN. For instance, if a file server is compromised

Any of the following, discussed in more detail in this section, may contribute to unauthorised LAN access:

- Identification and Authentication Mechanisms.
- Password Management.
- User Account Management.
- Workstation Security.
- LAN Device Security.

IDENTIFICATION AND AUTHENTICATION MECHANISMS

Identification and Authentication mechanisms regulate access to network services. Full or partial failure of these mechanisms may result in unauthorised access to network resources. Essentially, for any networking platform, these mechanisms are a first line of defence.

Identification and Authentication

NetWare uses a username and password combination to authenticate network logins and attachments. Each NetWare user account is recorded as a user object in the bindery of the server that the account resides on. Associated with bindery entry are a number of properties that describe the user object to NetWare, including a unique object identification number, the user name and a password for the object.

Identification and Authentication of users occurs during the login process. The user supplies a username and a password which is then checked against bindery information. Three possible events can occur during login:

- If a valid username and a correct password are entered, then the user is logged into the server.
- If a valid user name but an incorrect password is entered, the connection is refused and the server records the failed login. If the intruder detection account lockout feature is enabled, and the number of permitted password retries is exceeded, then the server disables the account for a pre-determined period of time.[3]
- If an invalid username is entered then the server requests a password as per normal, then refuses the connection. Note that this event is not logged in any way by the server.

Since attempts to log into the file server with an invalid username are not logged, the system administrator can only detect attacks that target existing usernames (either through the intruder detection lockout feature or through reviewing error logs for lockout records). The scale of an attack may be underestimated by an administrator, who has no way of monitoring the total number of failed login attempts.

PRE-LOGIN ACCESS TO NETWARE

Prior to the login process, a non-authenticated user is able to interact with the server in two ways: via the SYS:LOGIN directory and through the bindery of the target server.

The SYS:LOGIN Directory

In order to make programs such as LOGIN.EXE and SLIST.EXE available to client workstations, the server makes the SYS:LOGIN directory publicly available to all unauthenticated network clients. This directory should contain *only* files related to logging into the server or for getting information about other services available from the server bindery. Users are granted *READ* and *FILESCAN* rights to this directory, which allows them to list the contents of the directory and execute programs from the directory.[4]

Any program or data file that is placed in the SYS:LOGIN directory will be accessible by any unauthenticated user, and therefore care must be exercised when placing additional files there. Notwithstanding this, only privileged users are able to create or modify files in this

[3] The Intruder Detection Lockout feature of Netware is discussed in more detail on page 189 (User account Management).

[4] For a detailed discussion of file access rights, see "Access rights administration", on page 197.

directory, so it would be difficult for a viral infection or a Trojan horse program to take advantage of this directory. In addition, no other network directories are available to an unauthenticated user.[5]

Access to the server's bindery

An unauthenticated user is able to examine bindery objects that have a security level of ANYONE. This allows network clients to use the SLIST.EXE program to get server, router and network information. This access also allows a workstation to perform certain authentication oriented bindery activities, such as checking the password of a bindery object (using the `Verify_Bindery_Object_Password` call).

It is possible, using this access, to create a password cracker that checks a list of passwords against a user object in the bindery. Fortunately, the Intruder Detection lockout feature (discussed on page 189) provides protection from programs that use this kind of brute force attack, but there is still potential for "low priority" password crackers that slowly check passwords over a long period of time, while being careful not to invoke the lockout protection.

Another approach by an attacker might be to create multiple connections from a single workstation to the target server, from which a "parallel" attack could be mounted. This would still need to run as a low priority attack, but would complete substantially faster than an attack from a single connection.

THE LOGIN PROCESS

When the appropriate NIC and network shell drivers have been loaded at the workstation, the client is automatically connected to the nearest (or preferred) server, with a status of *NOT-LOGGED-IN*. This grants the station access to the servers SYS:LOGIN directory and certain bindery information available at the *ANYONE* security level.

On invoking the login program, LOGIN.EXE, the user enters the username and password of the account they wish to use. Optionally, they may specify the server that they wish to log into. In this case, the connected server locates and connects the login with the requested server. A step by step description of the login process is shown in Table B - 4.

---

[5] This may not always be the case, as a supervisor or supervisor equivalent user could unwittingly infect the LOGIN.EXE program with a virus under some circumstances. It is also conceivable that a trojan horse program could be planted here. These issues are discussed in more detail in section B.6 (page 212).

1. The login program at the client workstation asks the user for a username (U) and a password (P).

2. The client workstation issues a LOGOUT NCP request to the server, ensuring that any connection information from previous sessions is cleared.

3. The client WS requests from the server the *object ID* (U_{ID}) of the entered username and a *log-key* (L_K).

4. The client WS non-reversibly encrypts the [P, U_{ID}] pair to produce P_{16}, a 16 byte password value.

5. The client WS uses the same non-reversible algorithm and encrypts the [P_{16}, L_K] pair, producing a single 8 byte password value P_8, which is sent to the server for authentication.

6. The server authenticates P_8 by performing the same encryption process on the log-key that was given to the client and the password value (Bindery P_{16} value) stored at the server for this account. If the passwords match, then the user is authenticated for this server, and login is completed. If the passwords do not match, then the connection is refused and the lockout counter is incremented.

Table B - 4 The login process (adapted from Lamb[39]).

For versions of NetWare later than 2.15, at no time is a cleartext password value transmitted across the network. Unfortunately, If a pre 2.15 version of the network shell software is used by a client workstation, passwords will be transmitted in cleartext form. In this event, if the ALLOW UNENCRYPTED PASSWORDS file server console option is enabled, NetWare will allow the login to proceed. If this option is disabled, then an alert will be sent to the file server console and the request will be refused. As a matter of policy, the most recent versions of the Novell client networking software should be used on LAN workstations, and the ALLOW UNENCRYPTED PASSWORDS option should be disabled.

The encryption routines used for the login process, according to [39], are "not publicly known and are non-reversible". Passwords are stored within the bindery of the server in encrypted form, providing protection against passwords being revealed by perusal of a backup of the bindery. Note that it is not always the case that an encryption algorithm is secure simply because the encryption algorithm is proprietary and unpublished.[6]

[6] Recently, BYTE magazine published an article that included code to login to a netware server. This code implemented the "unknown" netware password encryption algorithm [45]. There are also various C implementations of the algorithm available, and Novell themselves make an object library available which will perform Netware compatible password encryptions.

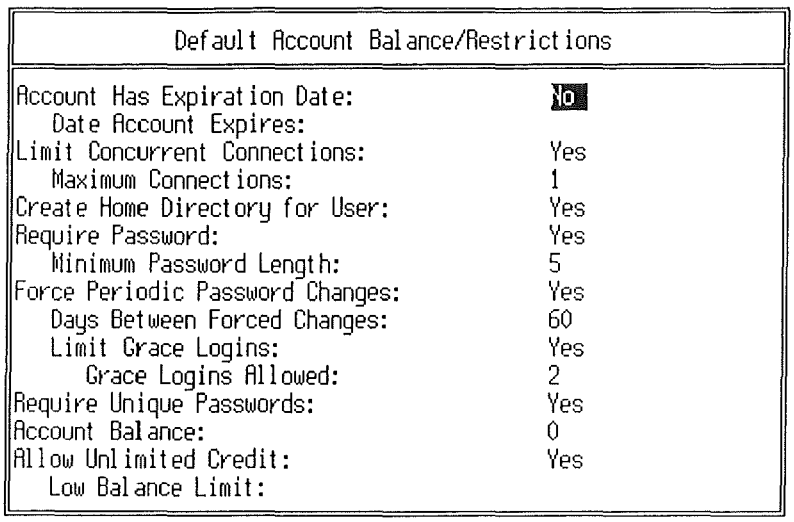


Figure B - 2 Default account settings (from the NetWare SYSCON utility).

A second non-public, but reversible, encryption system is used during the password changing process. This is used so that the server may decrypt a new password sent from the workstation. Because of the infrequency of password changes, this also does not present a major risk, although if a station is monitoring the LAN for password change sequences, and somehow knows the encryption system used, that station may be able to collect passwords for accounts as they are changed.

PASSWORD MANAGEMENT

The effectiveness of system authentication and identification mechanisms is very much linked to the strength of individual user passwords. Passwords that are too short, or are easily guessed, present a risk to overall system security. The risk becomes more acute with the increasing level of privileges that a user account owns. Proper password management is vital to ensure system security is maintained.

NetWare provides an extensive range of password options. In particular, NetWare system administrators need to be aware of the following issues (see Figure B - 2):

Requiring passwords

NetWare passwords are optional. By default, when an account is set up, no password is required. However, the system administrator is able to change the default settings so that password protection is mandatory.

Minimum password length

If mandatory passwords are specified, then the system administrator is able to set the minimum length of user passwords. Passwords that are too short (less than 5 characters)

are easily guessed, and NetWare allows password lengths to range from 1 to 20 characters. A setting of at least six is generally preferred, although this may vary according to local security policy guidelines.

NetWare does some minimalistic checking of passwords when they are set. For example, the system will not allow a password to be the same as the username of the account. However, in common with many systems, NetWare does not enforce good password choices, such as insisting on a mix of numbers and letters, or letters and punctuation marks. NetWare is also case insensitive with respect to passwords. For example, under NetWare, the passwords *SeSaMe* and *sesame* are identical.

Forcing periodic password changes

NetWare allows the system administrator to force users to change their passwords from time to time. This can be done on a per-user basis, or applied as a default to every user on a server. Additionally, the system administrator can specify the date that an account expires.

The time between password changes is expressed as a number of days. This defaults to 40 days, but the system administrator can specify a longer or shorter period. Users are also granted a number of "grace" logins, which allow them to continue using the system with an expired password. When the number of grace logins has been exceeded, the system forces the user to change their password anyway.

Requiring unique passwords

To prevent users from reusing old passwords when a password change is requested, NetWare remembers the previous ten password choices. When the *Require Unique Passwords* option is set, NetWare checks new passwords that a user enters against these previous ten passwords. If the new password has been used before, it is disallowed. Passwords need to be in effect for at least 24 hours before they are remembered by the system.

USER ACCOUNT MANAGEMENT

User account management is an important part of securing NetWare systems. These features allow the system administrator to limit the usage of user accounts to certain stations, times and numbers of connections. The system administrator is also able to set account expiry dates for accounts, minimising the risk of old accounts being reused to

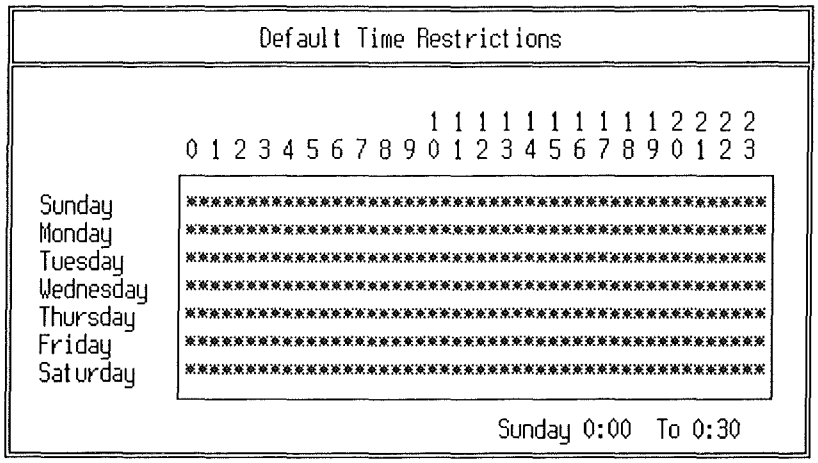


Figure B - 3 Default time restrictions (from the NetWare SYSCON utility).

attack the system. The features NetWare provides for user account management are discussed in more detail below.

Account expiry date

The system administrator is able to specify an expiry date for a user account. After this date, the account will be inaccessible to the user. The account may be re-instated by the system administrator by resetting the expiry date. This minimises the risk that old (and supposedly inactive) accounts are misused. This option is set on a per-user basis, and may not be applied to the SUPERVISOR account.

Limiting concurrent connections

By default, a user is able to open as many connections as they want. This is not always desirable, so NetWare allows the system administrator to define the maximum number of connections that a user is able to make to the file server. A limitation on the number of connections available to the SUPERVISOR account is possible.

Login time restrictions

As shown in Figure B - 3, NetWare allows the system administrator to specify the time of day that a user is able to log in and use the system. These times can be specified on an individual user basis or on a system wide default basis, and may be specified for each day of the week at half hour intervals. Login time restrictions are possible for the SUPERVISOR account.

Login station restrictions

Users can be associated with a physical station, by specifying the physical network address and the individual workstation node address that the user is allowed to login from. This is specified on a per user basis. Login station restrictions are applicable to the SUPERVISOR

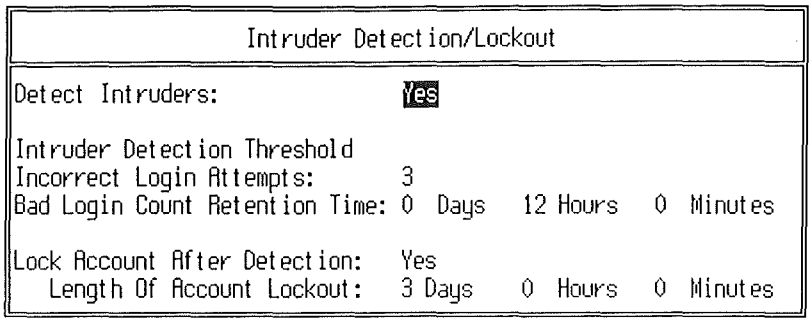


Figure B - 4 Intruder Detection Lockout settings (from the NetWare SYSCON utility).

account. Attempts to login out of the specified times or to login from an unspecified station are not recorded by NetWare.

Intruder detection lockout

NetWare includes an Intruder Detection Lockout feature (see Figure B - 4), which limit the number of login attempts that a user may make to an account. After the maximum number of login attempts is exceeded, the account is locked out for a pre-determined period of time. The system administrator is able to specify the number of login attempts that are possible, along with the period of time that bad login counts are retained. The Intruder Detection Lockout feature is set up on a default or individual user basis. The lockout feature may also be applied to the SUPERVISOR account.

Once an account has been locked, a record of the intrusion is logged in the `SYSTEM\SYS$LOG.ERR` file, and the account is flagged as inaccessible for the period of time specified in the Lockout setup menu. The account may only be manually reinstated by a system administrator.

The intruder detection feature may be used in a denial of service attack against a specific user account. Consider an attack on the SUPERVISOR account. If this is the only supervisor level account on the targeted system, and it is locked out by an attacker, then for the period of time that the lockout is effective, there can be no supervisory access to the server.

Even if there are SUPERVISOR equivalent users on the server, if these are known to an attacker, then they too are possible targets. If no supervisor equivalent users are left unattacked, then there are no privileged users left to unlock the attacked accounts. For this reason, it is wise to keep a secret, emergency supervisor equivalent user on the system.

The intruder lockout feature of NetWare also leaks information from failed login attempts, because the mechanism only applies to existing user accounts. An intruder can determine whether or not an account exists on the server simply by noting if the account gets locked out after a number of failed login attempts. This may be of some value to the intruder if the lockout time of the account is relatively small, ie. ten minutes or so, or if the attacker wishes to deny service to a certain account (for instance the SUPERVISOR account).

WORKSTATION SECURITY

Although the system administrator is able to verify the security status of centralised devices on the LAN, this often does not apply to user workstations, which are frequently outside the scope of administrative control. Various vulnerabilities attributable to user foibles and malicious attacks can erode system security. Most of these vulnerabilities can be attenuated to some extent by the provision of good physical security for network stations. Some specific vulnerabilities, for which NetWare provides little or no protection, include:

Workstation protection

Access control schemes applied to NetWare network resources do not apply once information is transferred to a user's local system. Unless that information is specifically protected, anyone who has physical access to the target workstation will have access to any information stored on that machine. Information that is sensitive in nature should be either stored on floppy's and physically secured, or should be encrypted on the local machine hard drive. Wherever possible, workstations should be protected by CMOS passwords which control who may boot the machine.

NetWare provides no in-built mechanisms to assist in securing workstations in any way. However, numerous third party solutions are available, both commercially and in the public domain.

Workstation session protection and automatic network logout

Users that leave themselves logged in while they are away from a workstation are also a potential system vulnerability. Once again, anyone who gains physical access to the workstation may use that user account, examining any information that the user account has access to, or propagating rights to their own or other accounts.

The seriousness of this problem increases with the increasing privileges of the accessed account. For example, if the SUPERVISOR or an equivalent account was left unattended, then for the period of time that the intruder is active, any activity is possible. The intruder

might choose to create a new SUPERVISOR equivalent account that he may access at his or her leisure.

The usual solution to this vulnerability is to implement software keyboard locking, in which users leaving their machines "lock" their keyboard with a password that must be typed before further access is permitted. NetWare provides no in-built mechanisms to provide this service, although most operating systems (for instance MS-Windows or OS/2) offer screen savers that provide password protection.

Another approach to this problem is to have the network server monitor user activity. If a workstation has been idle for some period of time (ie. 30 minutes), then it is automatically logged out. Once again, NetWare does not provide an idle workstation connection termination program, but numerous examples are available in the public domain.

Passwords stored in batch files

Users who store login passwords in batch files on their own machine, or who store passwords in login scripts also represent a risk to the security of the LAN. Those passwords may be compromised if physical access to the workstation is gained by an intruder.

NetWare does not provide services to counter this vulnerability, although physical security and education of end users about the dangers of storing passwords anywhere in their system is recommended.

Trojan horse login programs

Where NetWare workstations are shared, it is possible for a password collecting Trojan horse process to be loaded onto the machine. A Trojan horse of this nature will typically emulate the login prompts, fooling users into entering their user name and password. This information is then written somewhere that is secretly accessible to the perpetrator (ie. A hidden file) at some later time, and the real login process is then invoked, either with the captured username and password, or a message to the effect that the previous login attempt failed.

NetWare provides no built-in security features that prevent or detect Trojan horse attacks; as a matter of policy, users should always cold-boot shared and public workstations.

LAN DEVICE SECURITY

In order to fully secure the Network environment against unauthorised access, some thought must also be given to the protection of LAN devices. Improperly secured devices may result in a misuse of Network resources, or a denial of service.

Physical security is an important aspect of achieving LAN device security. Critical components, such as file servers must be physically secured, and if possible, password protected at the boot and console level.

File server console locking

NetWare provides a mechanism whereby the file server console may be password protected. This is invoked by starting the MONITOR program at the console, selecting **`Lock File Server Console`** from the main menu, and specifying a password to unlock the console. The console may be unlocked either with the specified password, or with the SUPERVISOR password. Note that attempts to access the console with incorrect passwords are not logged.

This feature, although useful, is also insecure due to a bug in NetWare. Any user who is a print queue operator can unlock the console by using PCONSOLE to down the print server running on the server that is locked. When this is done, not only is the print server task removed from the server, but MONITOR is unloaded, and the server is returned to the console command line.

The SECURE CONSOLE command

An intruder with physical access to a NetWare file server that has not been console locked is able to execute file server programs (NetWare Loadable Modules or NLM's) from either the file server hard drive's DOS partition (if it has one) or from a floppy disk. The SECURE CONSOLE command, if entered at the console, limits execution of NLM programs to those residing in the SYS:SYSTEM directory.

Additionally, this command prevents use of the keyboard OS debugger, prevents the date and time from being changed (important for auditing and accounting features) and removes DOS from the file server (preventing an exit to DOS, which would allow subsequent access to DOS programs at the file server).

Password protecting the file server hardware

If CMOS password protection is available for the file server machine, then it should be used. This will prevent an intruder booting from DOS and running a program that will steal data or alter file server information.

Protecting server boot disks and the SERVER.EXE program

File server boot floppy disks, if they are used, should be secured against physical access by anybody except the NetWare system administrator or console operator. If these are left unprotected, it is possible for an intruder to modify the SERVER.EXE program that is used to start NetWare 3.11 and 3.12 servers.

With the help of a DOS disk editor, an attacker is able to modify the SERVER.EXE program references to NET$OBJ.SYS, NET$VAL.SYS and NET$PROP.SYS (which contain the bindery) to other filenames. This removes any protection normally afforded by the bindery, and once this is accomplished, the attacker is able to login to the SUPERVISOR account without a password. If this occurs, the file server is left completely unprotected.

NetWare does not perform any integrity checks on the SERVER.EXE program file, and so an attack of this type is easy to execute if the perpetrator has physical access to a non boot-protected file server.

SECTION SUMMARY

In general, NetWare suffers from a lack of logging of security related events. As mentioned, NetWare only logs failed login attempts that are attempted on a specific account. *The system administrator has no way of knowing how extensively the network is being attacked.*

There are also specific vulnerabilities that are associated with physical access to NetWare file servers. As a rule of thumb, NetWare servers should have as much physical protection as possible, as they may represent a major security problem if left unsecured. This also applies to other network devices, such as routers, printers and modems. The vulnerabilities that may result if these devices are left unsecured are discussed later in this report.

In order to prevent cleartext passwords from being transmitted over the network, NetWare administrators should ensure that all users are using the latest versions of the NETX.EXE shell. Users should also be educated about good network security habits, such as not putting passwords in login scripts, not leaving themselves logged in at an unattended

workstation for long periods of time, and not sharing passwords and accounts with other users.

B.4 UNAUTHORISED ACCESS TO LAN RESOURCES

Unauthorised access to LAN resources occurs when a legitimate or unauthorised user gains access to LAN resources that he or she should not otherwise be entitled to. This threat can be the result of improperly assigned access controls, or can be due to insufficiently granular access controls.

The availability of NetWare LAN resources is influenced by three main factors: user security levels, access rights administration, and the NetWare default system configuration.

User security levels determine what type of privileges a user is entitled to, while access rights determine which users are allowed access to which resources. The default system configuration can influence the amount and type of access that users have. The following subsections discuss these factors in detail, along with the security services that NetWare provides to control access to LAN resources.

USER SECURITY LEVELS

Limiting the activities of users is fundamental to maintaining good security in a networked environment. In order for the LAN to be administered, it is necessary to allow some users a greater variety of privileges than others. However, mismanagement or misuse of privileged user accounts can lead to vulnerabilities.

NetWare supports the concept of *End Users*, who are subject to normal access controls, and *Administrators*, who are granted additional privileges in order to carry out specific system management functions. Of significance to NetWare security is that access control mechanisms (described in the next subsection) respond differently according to the security level of the user. Figure B - 5 shows the relationships that exist between users and administrators.

The end user

End Users are the normal, unprivileged user accounts that constitute the bulk of users on any NetWare server. These users are subject to the default access control mechanisms enforced by NetWare. End users may be granted additional privileges by being assigned to groups, or by having additional rights assigned directly to them.

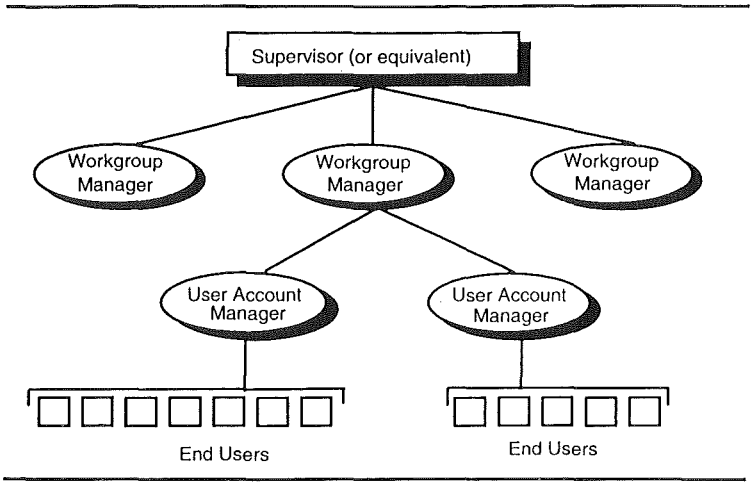


Figure B - 5 User security level structure (adapted from Sheldon [71]).

Security equivalence

Any user account is able to gain additional privileges via a mechanism known as *security equivalence*. Security equivalence is a one-way relationship from one user to another. If user REN is security equivalent to user BOB, then REN has access to all information (and all the rights) that BOB has access to. BOB, on the other hand, does not have access to all REN's information. This relationship mechanism is useful for giving a user *temporary* access to files that another user has.

Use of security equivalence should be used sparingly, as carefully thought out access control schemes may be thwarted by users gaining rights they should not be allowed via an obscure security equivalence. In a sense, equivalence is a convenient shortcut that can nullify most of NetWare's access controls mechanisms.

NetWare administrators

There are five different types of NetWare administrator, which fall into two categories; system administrators and user administrators. A system administrator, such as the SUPERVISOR or a supervisor equivalent user, is able to change NetWare system parameters, delete and create user and group accounts and assign overall rights to other users and administrators. A user administrator, such as a workgroup manager, carries the responsibility of managing end users.

System administrator (SUPERVISOR)

The SUPERVISOR account owns all rights to a NetWare system, and may make changes to object attributes. A System administrator may assign user account manager, workgroup manager and operator privileges to an End User. The SUPERVISOR account may not be removed from a NetWare system.

As NetWare does not allow the removal of this account, it is an obvious target for attack. NetWare system administrators should restrict use of the SUPERVISOR account to known, physically secured workstations (using the account restrictions discussed earlier in this report), and should limit the times that the account can be used. Additionally, limiting the number of concurrent connections to this account is a good idea.

System administrator (SUPERVISOR equivalent)

This is a user account granted SUPERVISOR level rights, with the NetWare equivalence function. This type of account has all the rights of the SUPERVISOR, and is able to propagate those rights to other user accounts. Supervisor equivalent accounts may delete this attribute from other supervisor equivalent accounts, but may not revoke supervisory rights from the original SUPERVISOR account.

The SUPERVISOR and supervisor equivalent accounts are the most powerful accounts found on a NetWare server, and use of these accounts should be monitored closely. In particular, administrators should regularly run the SECURITY.EXE program provided with NetWare to ensure that only authorised users have the supervisor equivalence property set[7].

User account manager

A User Account Manager is an administrator that has supervisory level privileges for all user and group accounts assigned to them. This manager can set account balances and user passwords, can create and assign users to groups, and can change security equivalence. Additionally, the User Account Manager can assign user and group trustee rights for resources to which they themselves have access. User Account Managers are able to delete accounts that they have created.

Workgroup manager

A Workgroup manager is able to perform the same tasks as a User Account Manager, and is additionally able to create new user accounts and create, manage and delete print queues. Workgroup managers are able to delete accounts assigned to them or created by them.

[7]The SECURITY.EXE program is discussed later in this report.

Operator

An Operator is an End User that is granted special access privileges in two areas: FCONSOLE and RCONSOLE operation, which allows the operator access to remote file server console operation; and PCONSOLE operation, which grants the user special rights regarding the operation of NetWare print servers.

NetWare administrators should carefully monitor who gets RCONSOLE and FCONSOLE access, as these privileges allow that user direct access to file server control functions.

User groups

End users and administrators can be made members of one or more User Groups. Directory access and trustee rights can then be assigned to groups of users, simplifying the administration process. By default, all users are added to the NetWare default group EVERYONE.

The default user GUEST is installed with NetWare. This account is assigned to the group EVERYONE, and therefore has access to all the resources that this group has. It is recommended that this account be removed from the system, as it is another obvious target for attack.

ACCESS RIGHTS ADMINISTRATION

Access control management is necessary to prevent unauthorised access to LAN resources. The goal of access control mechanisms is to limit the resources that a user can access once they have logged in to only those files and directories that they need to use.

NetWare controls access to files and directories by making use of two complementary sets of access control information: The *rights* that have been assigned to the user to access the files and directories; and the *attributes* that have been assigned to the files and directories themselves.

Directory and file rights

Directory rights control general access to directories and files, and may be applied to either. NetWare uses a total of eight rights flags to determine the information a user can access, and what the user is able to do with that information. Both the trustee assignment and the Inherited Rights Mask mechanisms use these rights for access control. The rights, and their effects when applied to either a directory or file, are shown below in Table B - 5 on the next page.

Right	Applied to Directory	Applied to File
Supervisory	Grants all rights to the directory, its files and its subdirectories. This right overrides any Inherited Rights Masks restrictions that may be placed on subdirectories, and may not be revoked at a lower level in the directory tree	Grants all rights to the file. Users with this right may grant others access to the file, and can modify all rights in the file's Inherited Rights Mask
Read	Grants the right to open files in the directory, read the contents of those files, and execute programs in the directory.	Grants the right to open and read the file.
Write	Grants the right to open and modify files in the directory.	Grants the right to open and modify the file.
Create	Grants the right to create files and subdirectories in this directory. Can be used to create a drop box directory that others can create and open files in but cannot read from. (ie mail directories). The C right automatically grants the W right.	Grants the right to salvage the file after the file has been deleted.
Erase	Grants the right to delete the directory, its files and its subdirectory files.	Grants the right to delete the file.
Modify	Grants the right to change the directory and file attributes of this directory, to rename the directory or its files and subdirectories. Does not grant the right to change the contents of the file.	Grants the rights to change the files attributes and the file name, but does not grant the right to modify the contents of the file.
File Scan	Grants the right to see directory entries.	Grants the right to see the file in a directory listing. Grants the right to see the directory tree path from the file to the root of the tree.
Access Control	Grants the right to modify the directory trustee assignments and inherited rights mask. Users can grant any right (except Supervisory) to any user, even rights that they do not themselves have.	Grants the right to modify the file's trustee assignments and inherited rights mask. Users can grant any right (except Supervisory) to any user, even rights that they do not themselves have.

Table B - 5 NetWare file and directory trustee rights (from the NetWare Concepts manual).

Determining effective rights within a directory

The *effective rights* a user has in a directory are controlled by the rights that the user inherits from the parent directory, modified by the *inherited rights mask*, and by the set of *trustee rights* the user has for the directory.

Trustee rights assignments grant a specific user or group the right to use a file or a directory in a particular way. Trustee assignments are generally carried out by an administrator, but may be carried out by end users if they have the Access control right.

A trustee assignment in a subdirectory automatically lets the user see the directory tree all the way back to the root directory. Other subdirectories may only be seen if the user has other trustee assignments in the directory tree.

Trustee rights to a directory may come from an assignment made directly to the user, a trustee assignment made to one of the groups that the user belongs to, or from a trustee right assigned to another user with whom the user is security equivalent.

Inherited rights masks control what rights a file or directory inherits from its parent directory. The default is all rights, but if this can be modified so that certain rights present in the parent directory are revoked at the current level.

Inherited rights masks are applied directly to a directory or file. They can be used to revoke any or all of the eight file access rights shown in Table B - 5, except the supervisory right. If the supervisory right is effective in a parent directory, *then that right may not be revoked in any subdirectory using the inherited rights mask mechanism.* This can be a particularly important consideration when granting a user supervisory trustee rights at a high level within a directory structure.

If there is a trustee assignment for a directory, then that trustee assignment overrides the rights that would otherwise be inherited by the parent directory, *regardless of the Inherited rights mask.*

The algorithm for determining the effective rights that a user has to a directory is summarised in Figure B - 6 on the next page. Note that if a SUPERVISOR or equivalent user has the Supervisory right (S) in a directory, then *all* access rights from that directory down are granted. This has implications for administering the LAN, as this right can not be revoked at a lower level, even with a new trustee assignment. It must be revoked at the same level that it was granted. Administrators must be sure that they do not grant this right at too high a level in the directory tree, as otherwise the user might gain too much access to the rest of the file system, in unauthorised access vulnerabilities.

As the S right also implies access control, any rights access (except supervisory) may be propagated to other accounts. This may not always be desirable, as it could undermine carefully thought out access control schemes.

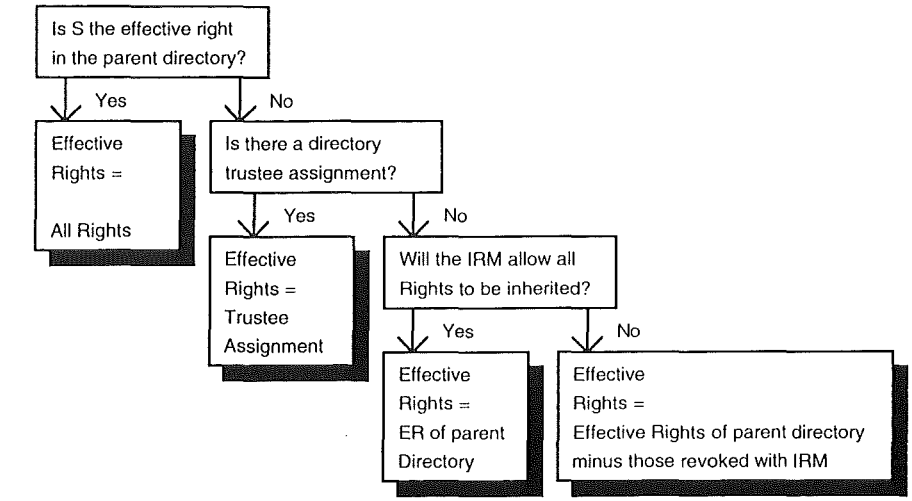


Figure B - 6 Determining effective directory rights for NetWare (from the NetWare Concepts manual).

Directory and file attributes

NetWare security attributes are applied to files and directories, and are enforced regardless of the security level of the subject that requests access. There are three types of attributes in NetWare:

- *File Attributes*, which apply to files within a directory.
- *Directory Attributes*, which apply to the directory itself.
- *Directory Rights* (the Inherited Rights Mask) assigned to the directory[8].

NetWare uses file attributes to assign special properties to files or directories. Attributes override effective rights and can be used prevent tasks that those rights might otherwise allow. The use of attributes increases the level of access control granularity available to the administrator; attributes are also useful in preventing certain types of viral infections. The Modify effective right is required to change directory or file attributes.

Attributes for each object are indicated by the presence or absence of the flags shown in Table B - 6. Most of the flags apply only to files; those applicable to directories are indicated as such in the Directory column.

[8]The Inherited Rights Mask dictates access allowed within the directory relative to access rights in the parent directory. Because it is applied irrespective of the rights of the user, the IRM can be viewed as an *attribute* of a file or directory.

Attribute	Directory	Description
Archive Needed		Assigned automatically by NetWare, and identifies files modified since the last backup.
Copy Inhibit		Prevents Macintosh users from copying the file. Overrides read and filescan rights. The Modify right is required to remove this attribute.
Delete Inhibit	Y	Prevents users from erasing directories and files. Modify right required to remove this attribute.
Execute Only		Prevents copying or backing up of the file. Attribute can never be removed. This attribute should only be assigned to executable's, and a backup of the original should be kept. Can only be assigned by a supervisor.
Hidden	Y	Hides directories or files from a DOS DIR file listing, and prevents them from being deleted or copied. Will be listed with NetWare's NDIR command if the user has the appropriate filescan rights.
Indexed		Allows large files to be accessed quickly, and is automatically applied to files that have more than 64 regular FAT entries. Can be set, but has no effect.
Purge	Y	Purges a file as soon as it is deleted if it has this flag or resides in a directory with this flag. Purged files may not be recovered with NetWare's SALVAGE command.
Read Audit		Not used. May be set but has no effect.
Read Only/ Read Write		Indicates whether the file can be modified. All files are automatically flagged as Rw on creation, and may be modified if the Ro attribute is not set. Assigning Ro automatically assigns Delete Inhibit and Rename Inhibit. The Modify right is required to remove the Ro attribute.
Rename Inhibit	Y	Prevents user from renaming directories and files. Modify right required to remove this attribute.
Shareable		Allows multiple users simultaneous access to the file. Modify right required to remove.
System	Y	Assigned to system files and directories. Hides flagged files from a DOS DIR scan and prevents deletion and copying. Directories and files will appear in the NDIR listing if the right filescan rights are present.
Transactional		Activates the Transaction Tracking System. All changes to a file with this attribute set will either be executed to completion, or will fail with no changes to the file. Used for database files.
Write Audit		Not used. May be set but has no effect.

Table B - 6 NetWare file and directory attributes (from the NetWare Concepts manual).

Rights Required for user directory/file operations

In order to manipulate directories and files, the user needs to have both the correct effective rights for the objects that are to be manipulated, and the necessary attributes cleared. The operations that are possible, along with the effective rights that the user must have to perform these operations, and the attributes that affect those operations are summarised in Table B - 7 on the next page.

Operation	Effective Rights required for the target directory or file.	Attributes involved.
Copy a file into a directory.	Create right.	None.
Create and write to the opened file.	Create right.	None. Rw (Read Write) is assigned to the file when it is first created.
Read from a closed file.	Read right	None.
Modify disk space assignments in subdirectories.	Supervisory right.	None.
See the root directory	Any right in a subdirectory.	None.
Copy a file from a directory	Read and filescan rights.	System attribute must be cleared, and if the user is a Macintosh user, then the Copy Inhibit flag must be clear.
Delete a file.	Erase right	Delete Inhibit must not be set. Delete Inhibit automatically assigns Read only and Rename inhibit attributes. Neither of these attributes must be flagged for the deletion to work.
Remove an empty subdirectory.	Erase right.	Delete Inhibit flag must be clear.
Write to a closed file	Write, Create, Erase, Modify	The file must be flagged as Rw (Read Write).
Rename a file or directory.	Modify right.	None.
Search for files in a directory.	File Scan right.	Files may be hidden from a DOS DIR scan with the System or Hidden attributes
Search for subdirectories in a directory.	File Scan right.	Directories may be hidden from a DOS scan with the System or Hidden attributes.
Change directory or file attributes.	Modify right.	None.
Change trustee assignments.	Access control right.	None.
Change the IRM.	Access control right.	None.

Table B - 7 Rights and attributes required for user operations.

Rights required for queue access

Access controls are necessary to control who is allowed to utilise and administer NetWare print queues and job service processes. The rights to create and grant access to queues are limited to NetWare administrators. The system supervisor is automatically made a queue operator of any new queues that are created.

NetWare allows users or groups of users to be granted Queue User, or Queue Operator effective rights. The Queue Server right must be explicitly granted by the supervisor to a specific server process. The effects of assigning a user rights for a particular queue are detailed below.

- *Queue User Right.*
  This Right allows the user to submit jobs to the queue it is effective for, to view to queue's job list and to view the status of the queue. This right allows the user to cancel any jobs that they themselves have submitted to the queue. The group EVERYONE is automatically assigned Queue User rights to all print queues, although this may be revoked.

- *Queue Operator Right.*
  This grants the user special rights regarding the operation of queues for which it is effective. The right allows the user assign other queue operators, edit queue entry requests, change the order of jobs to be serviced, delete entries from the queue and modify the status of the queue.

- *Queue Server Right.*
  The queue server right is granted to specific queue *servers*. A server is either a user starting a service process on a workstation, or the server process itself when it logs into the file server. This right grants server processes access to jobs in queues. Note that in order to service a particular job, the queue server will sometimes need to inherit security equivalence of the users whose job is being serviced, and thus the service process's effective access rights may be temporarily increased in the course of servicing jobs.

NETWARE DEFAULT SYSTEM CONFIGURATION

The default NetWare directories `SYS:LOGIN`, `SYS:SYSTEM`, `SYS:MAIL` and `SYS:PUBLIC` are set up at installation time. Access rights to these directories should be carefully controlled, as giving users too much access to the system or public directories could lead to vulnerabilities that result in unauthorised LAN access. The default installation directories, and the recommended maximum rights granted to users in those directories, are shown in Figure B - 7 on the following page.

The SYSTEM directory

NetWare stores all its security sensitive utilities, its NLM modules and its auditing log files in the `SYSTEM` directory. For this reason, *no users apart from supervisory level users should be allowed access to this directory.*

The LOGIN directory

All users should have all rights to the `LOGIN` directory revoked, as once they have logged into the server there is no reason for them to access this directory. This reduces the risk of

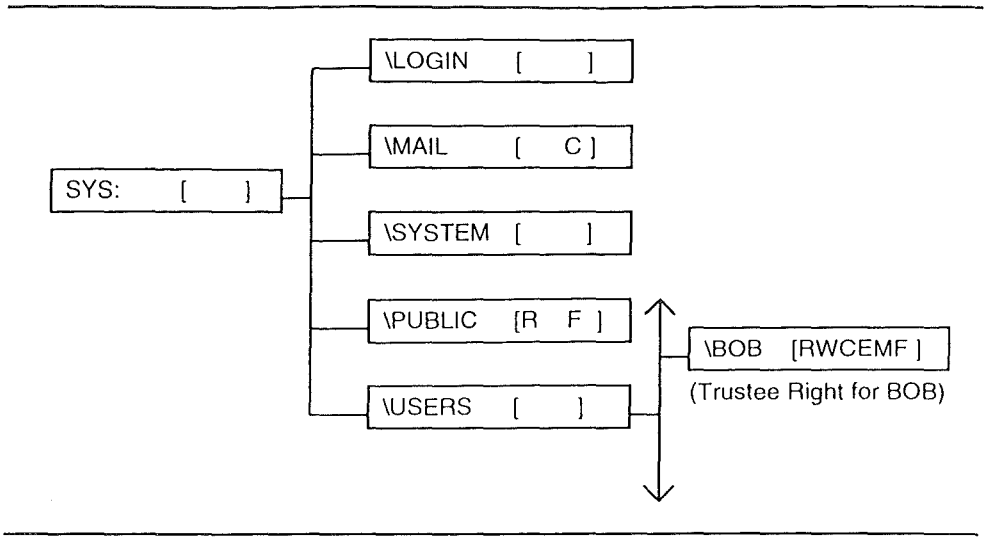


Figure B - 7 A default NetWare directory configuration (the USERS directory has been added subsequent to the original NetWare installation).

a Trojan horse attack initiated by placing a subversive LOGIN.EXE program in this directory. Only Supervisory access to this directory should be allowed.

A viral infection of files in the LOGIN directory is still possible. See section B.6, Unauthorised Modification to Data and Software, for more details.

The MAIL directory

The MAIL directory is not only used for mail, it is also used for storing individual user login scripts. When a user is created, a subdirectory in the MAIL directory is assigned to that user by the NetWare system, and is named using the user's bindery object ID. The user login script (stored in the *login* file) is placed by NetWare in this directory.

By default, all users are granted Create rights in the MAIL directory, which has the effect of creating a "drop box" directory. Any user is able to open a file in another user's mail directory, and while that file is open, is able to write information such as a mail message, to it. A well-known NetWare vulnerability can result from this setup:

NetWare administrators should ensure that a user has a login script in the file *login*, even if it is just one character long (ie. a space). Every user has the create right in the mail directories; If a user does not have a login file, then anyone can create a file called *login* in that user's mail directory. This login script would execute next time the victim logged in, and could contain commands that steal information or grant the perpetrator additional rights to see that user's files. This is especially important for users that may have supervisory level privileges that are able to be propagated to other users.

The PUBLIC directory

Files and utilities that all users require in order to use the NetWare system are placed in the PUBLIC directory. Users should have all Trustee rights revoked to this directory, and the group EVERYONE should be given read and filescan rights. This will allow users the ability to execute the utilities, but will prevent them from deleting or modifying them. The NET$LOG.DAT file which stores the system login script is also placed in the PUBLIC directory. This file should be protected from modification by any user except a system administrator.

The USERS directory

This directory, shown as an example in Figure B - 7, is not specifically part of the NetWare default setup. It is, however, a recommended setup of user home directories. If the system administrator wishes to prevent users from granting access to files in their home directory to other users, then a Trustee right of all rights except the Access control and Supervisory right is necessary for that user's home directory.

If copying of files between users is permitted, then additionally granting the Access control right to that user in that directory is required. In the example given in Figure B - 7, user BOB is able to use his files normally, but is prevented from granting access to those files to any other user.

Application directories

Granting users more access than they need to some directories is sometimes unavoidable with NetWare. This is mainly due to certain applications that try to open files in the same directory that they reside in. In cases such as these, setting the directory rights to only Read and Filescan will cause the programs to fail. For this reason, some applications will need to reside in directories with nominally more access rights available than should really be needed.

In these directories, program files should be protected, if possible, with the execute only attribute, while read only data files should be flagged as such. If the program needs read/write access to a file, then granting such access to the program also effectively grants users access to that same file. This could create a security problem if the user makes some unauthorised modification to the program's data. NetWare does not have a Unix-like "set

user id" mechanism that can be used to overcome this limitation, except in the operation of print queues.[9]

Unfortunately, setting the Execute only attribute on program files may also cause certain programs to fail. Where possible, these applications should be identified and isolated as much as possible from the rest of the system.

SECTION SUMMARY

During tests of NetWare carried out by Cohen [14], it was found that by far the most pressing weakness in NetWare security is the complexity of the access control mechanisms. By way of illustration, to access a file in a directory, NetWare needs to consider 14 file attributes, 8 file trustee rights, 8 inheritance rights and 16 rights (8 inherited and 8 trustee) per parent directory back to the root directory .

Changing one bit of protection at a high level in the directory structure can have major effects lower down in the directory tree. Similarly, granting a user any right in the root directory of a volume effectively grants that user the same right to the entire volume. Granting the Access control right is especially risky, as this allows the user to grant themselves any right for the entire volume.

Additionally, users may have security equivalence's that grant access rights to files and directories in unpredictable ways. Administrators using the security equivalence function may unintentionally grant a user access to resource's that they should not have access to. As stated, equivalence should be used sparingly.

Another potential problem, discussed in [14], is that NetWare sometimes takes a while to make rights effective (most likely because of the complexity of the mechanism), and as a result windows of vulnerability may exist while rights changes are made. This may impact some MS-DOS programs, which are normally used in a more static environment, and could fail to take these (unpredictable) time lags into account.

In NetWare's favour, access controls are very granular. User access rights can be restricted to individual files within a given directory, and attributes are set on a per-file basis.

[9] Of course, SUID script files and programs are a major source of Unix system vulnerability.

The SECURITY.EXE program

The SECURITY.EXE program provided with NetWare can assist system administrators in finding a number of potential holes. When run, this program checks the bindery of the server for the following potential problems:

- *Users with no password.*
  A user account with no password is an obvious security problem, as anyone will be able to login to that account without authorisation.
- *Users with insecure passwords.*
  Users with passwords that are the same as their login account name, or with passwords that are too short are potential security problems. The SECURITY.EXE program will also check for user passwords that do not expire at least every sixty days, and for accounts that do not require a unique password.
- *Users that are security equivalent to the SUPERVISOR.*
  The number of SUPERVISOR equivalent users should be kept to a minimum. The SECURITY.EXE program checks for users that are security equivalent to the supervisor.
- *Users that have privileges in the root directory of a volume*
  As stated, granting users privileges to the root directory of a volume can lead to vulnerabilities caused by the propagation of that right to the entire volume.
- *Users with excessive rights in the standard directories.*
  SECURITY.EXE will check to ensure that no users are granted more than the minimum set of rights recommended for the four system installed directories (as shown in Figure B - 7).

B.5 COMPROMISE OF DATA AND SOFTWARE

Compromise of LAN data and software occurs when an individual breaks the confidentiality of data that they should not be privy to by accessing and comprehending that data. In addition, the software itself may be compromised by an attacker who seeks to take copies, or to search executables for sensitive data stored as part of the program.

The factors that can influence compromise of data and software include:

- File system encryption services.
- Access controls.
- Object reuse.

- Physical placement of output devices.
- Security of system backups.

FILE SYSTEM ENCRYPTION SERVICES

Encryption of sensitive information stored on network devices is essential if that data and software is to remain confidential. Encryption services should be used in any situation where there is sensitive or confidential data stored on the LAN.

NetWare provides no file system encryption facilities, apart from those associated with the transmission of passwords. Hence, if data is to be encrypted, then a third party encryption package is required.

ACCESS CONTROLS

The role of access controls in regulating access to confidential information is to keep those users who do not require access to that information from gaining access to it. Unfortunately, access control mechanisms themselves are subject to compromise, and should be seen as only a first line of defence. For a variety of reasons, discussed below, NetWare access controls may fail to prevent unauthorised perusal of confidential data:

- *Improper Access control configuration*
  NetWare access control mechanisms, as discussed on page 206, are complex to set up. Administrators could inadvertently assign the wrong set of access rights, or could fail to take a security equivalence into account when assigning rights.
- *Malfeasance*
  NetWare system administrators, such as the SUPERVISOR or an equivalent, are not subject to the access controls that dictate end user rights on file server volumes. A supervisor is able to access any data stored on the server they are a supervisor for. If that data is confidential, and the supervisor is a person who is normally not authorised to access that data, and if the data is unencrypted, then that data has been compromised. NetWare is particularly vulnerable to malfeasance attacks from supervisory users abusing their privileges.
- *External attacks*
  An intruder that gains access to a user account, or even worse to a supervisory account, is able to access any unencrypted data that the account is able to access.

- *No access controls at local workstations*
  access control mechanisms may prevent compromise of data and software that is stored on a NetWare server, but once this data is transmitted to a local workstation, then that data has no access control protection at all. Any intruder who gains access to the local workstation is able to compromise that data if it is unencrypted.

OBJECT REUSE

Object reuse occurs when some item of data is, either through chance or design, accidentally included as part of the data set of some other object when that object is created or changed. This type of information compromise is often unpredictable, and usually is the result of improper initialisation of areas of memory before that memory is reused.

For NetWare, object reuse concerns three main areas: file server disk storage, file server communications buffers and network interface card buffers [39]. In addition, some applications accidentally compromise information through internal buffer reuse. Each of these areas are covered in turn:

- *File Server Disk Storage*
  When a user requests an allocation of disk space, and does not provide data for that space, NetWare pre-zeros the space allocated before giving access to that disk space to the user. This has the effect of ensuring that information that was previously stored on the allocated space is not available.

- *Communication Buffers*
  NetWare servers process incoming and outgoing requests in separate buffers. If a reply includes data from the file system, then that data is directly packetised from the file system cache buffers (ie direct from the server disk volume). NetWare maintains a connection slot for each user. If the connection slot is re-used (ie one user logs out and another logs in and acquires the same connection slot), the fields of the connection slot are zeroed before the connection is reallocated to the new user of the slot.

- *Network Interface Card buffers*
  In some cases, if a packet is generated by either the server or a workstation, and is less than the minimum packet size of the network topology in use, then the remainder of the packet may be padded out with space from the network interface card's data buffer. This can result in previous information used by the card being included in another packet, leading to compromise of the that data.

- *Application buffer reuse*

  It is possible that an application can accidentally leak information if its internal buffers are reused. Again, this results from a client application re-using areas of allocated memory that were previously used to hold some important item of data.

An example of application buffer reuse was discovered during testing of NetWare 3.12. In the scenario encountered, a password protected object was created, a password specified, and a full name (NetWare's IDENTIFICATION property) of less than 35 characters specified. If carried out in this sequence, these activities cause the password text to appear in the IDENTIFICATION buffer (after the full name zero terminator). This information is retrievable by any user with a bindery examination tool.

It appears that the applications involved (SYSCON and PCONSOLE) reuse an internal buffer, which accidentally includes the password text. Initially it was hypothesised that this reuse may be due to network interface card buffer reuse, but further investigation with a packet sniffing utility suggested that the fault probably lay with the Novell supplied applications rather than the network interface card and driver layer.

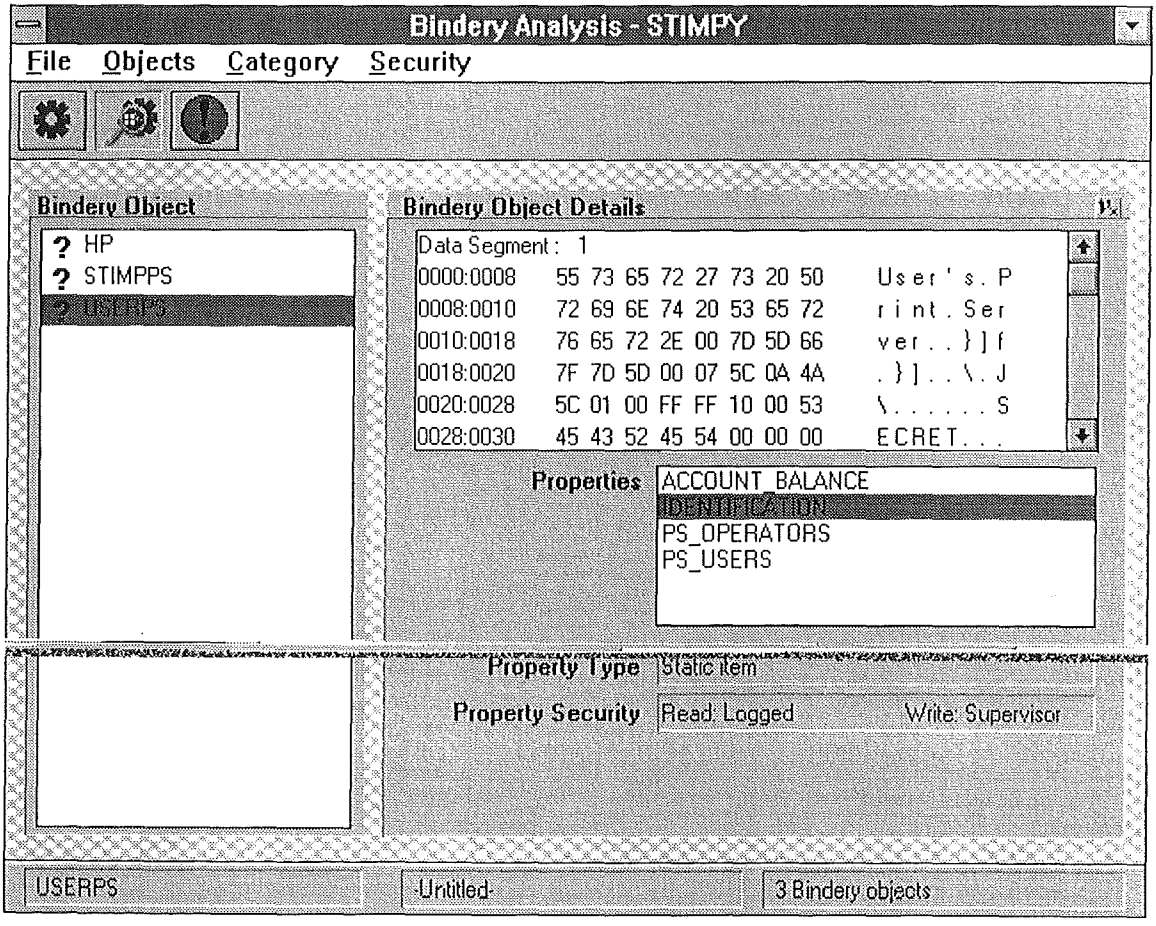


Figure B - 8 NetAudit Screen showing bindery object reuse for object USERPS.

This reuse problem is illustrated in Figure B - 8. This diagram shows a snapshot of the print server bindery object USERPS immediately after its creation. The object was created with PCONSOLE, and the password ("SECRET") and full name ("User's Print Server") were specified immediately afterwards.

The password text can be clearly seen at offset (hex) 27. Setting the full name field immediately after setting the password generally always produces the same result in both PCONSOLE and SYSCON. Once the full name is specified again, the password text is overwritten.

PHYSICAL PLACEMENT OF OUTPUT DEVICES

Sending sensitive or confidential information to a public printer may result in that information being compromised. Likewise, outputting the data to a screen that can clearly be viewed by an unauthorised person is also a vulnerability. The main way of preventing this type of compromise is to provide a physically secured area for printers that output sensitive information, and to provide physical shielding of screens that may display sensitive information.

NetWare allows printers to be placed anywhere local or remote to the server, so physical protection of output should not present a problem.

PHYSICALLY SECURING BACKUP INFORMATION

Data and software may be compromised if an unauthorised party gains access to backup tapes or disks. NetWare does not encrypt backup information, so if physical access is gained to these backups, then a compromise of any confidential information stored on those backups may be the result.

SECTION SUMMARY

The main weakness of NetWare, as discussed above, is the lack of encryption services for data, both stored on the file server, and on local workstations. Add to this the potential problem of supervisory malfeasance, the complexities of assigning rights and the potential for compromise of backups, and NetWare becomes quite vulnerable to this threat. These problems can be alleviated to some extent by the installation of encryption software.

B.6 UNAUTHORISED MODIFICATION OF DATA AND SOFTWARE

Unauthorised modification of Data and Software occurs when programs or files are subjected to unauthorised modifications, deletions or additions. Three main controls can reduce the likelihood of this threat:

- Access Controls.
- Detection of changes to system binaries and applications.
- Integrity checking of data files.

By far the greatest threat in this area comes from Viral infections. MS-DOS has a number of vulnerabilities in this area, and when files on a server become infected, especially those in public areas, the spread of the infection can be rapid.

Other changes, if undetected, can cause software corruptions, can divert information to unauthorised destinations and can result in incorrect information being produced by systems that run on the LAN.

ACCESS CONTROLS

Access controls can be applied to program or data files to reduce the risk that unauthorised modifications will be made. Access controls may fail for any of the reasons given on page 206, or because of some of the considerations given below. Again, for these reasons access controls should be considered a first line of defence in the prevention of unauthorised modifications. A proper regime of detecting changes to application files and binaries (discussed later) should also be in place.

The execute-only flag

NetWare allows the system administrator to restrict changes to executable files (.EXE or .COM files) via the use of the execute-only file attribute. This attribute restricts all access to execution of the program file only. The contents of the file are not able to be inspected, the file can not be written to, and the execute-only attribute can not be cleared, even by the system administrator. Note that with this attribute set, some programs will not run.

The execute-only attribute by itself is inadequate for protection against so called companion viruses. A companion virus works by replacing the original program file with one that performs a different, possibly malicious task. To prevent detection by a comparison of file sizes, the companion virus is given the same file size and other external characteristics as the original file.

With NetWare, an execute-only program file can be renamed, and a companion virus substituted and flagged as execute-only. *The only way to prevent this type of infection is to flag the original program file with the rename-inhibit attribute in conjunction with the execute only attribute.* [14]

Although the execute-only attribute is useful for preventing modifications to program files, its use presents another problem. *Files that are flagged execute-only can not be scanned for changes, as this attribute prevents normal read access of the file.*

In general, users should not be permitted write access to directories containing program executables. Because the modify right allows the user (and consequently a virus) the ability to trivially change permission's, this right should also be revoked in these directories. *Denial of write rights in a directory should always be accompanied by a denial of the modify right.*

The SUPERVISOR account and unauthorised modifications

The NetWare SUPERVISOR account, is not subject to normal access control mechanisms; rather, this account has *all* rights in all directories, including the ability to write to files and modify attributes.

For these reasons, if a SUPERVISOR user ever logs in from an infected machine, then the virus that has infected his machine *automatically has access to the entire file server.* The most obvious immediate infection would be of the LOGIN.EXE program. If this is infected, then all users logging into the file server would be infected.

Extra care should be taken to ensure that Supervisory accounts never *log in to a NetWare server from an infected machine.*

DETECTION OF CHANGES TO PROGRAM FILES

Viral infections, Trojan horses and unauthorised (or unintentional) user patching of software fall into this category of vulnerability. In recognition of the fact that access controls may not be able to prevent all unauthorised modifications, system administrators should install software that helps detect any changes that have been made to critical files.

Although NetWare does not provide either built-in detection of viral activities or automatic change control management, standard methods exist whereby changes and modifications are able to be detected. The most useful of these are Virus detection scanners, Cryptographic checksumming of executables, and third party packages that detect changes to critical files.

Virus detection scanners

System administrators of NetWare networks should have a policy regarding the compulsory use of virus detection software at user workstations. A convenient method of enforcing this policy is to place a memory resident (TSR) virus scanner in the system login script. This program will then be automatically run as part of the login process. The main goal of this protection is to minimise the threat of viral infections spreading from user workstations onto the file server. Note that some TSR virus protection programs will interfere with the normal operation of the workstation.

Virus detection software should be used to periodically to scan all files on the file server. Some commercial virus detection packages that run as NLM's (NetWare Loadable Modules) on a NetWare server are available (see Salamone [66]).

Cryptographic checksums

Another method that may be used to increase executable file integrity is to encase programs in a Cryptographic checksum system. This works by using a secret key to calculate a cryptographically secure checksum of the contents of the executable. A shell is added to the program that is activated when the program is run. It requests the key and then re-calculates the checksum and if the stored value does not match the original, then further execution of the program is cancelled.

This may affect the normal operation of programs, and can not easily be used in conjunction with the execute-only attribute.

Change detection

System administrators should periodically check file information, such as size, last modification date and attributes. This applies particularly to security sensitive areas of the file server, such the SYSTEM, LOGIN and PUBLIC directories, where changes should only be made by an administrator.

Several third party change management packages are available to track changes to NetWare files. Where possible, a package should be selected that uses some form of cryptographical checksumming technique so that changes to the contents of files can be detected.

The effects of viral infections on user workstations

Some work has been carried out to determine the effects that virus infections have on the NetWare IPX and NETX programs, most notably by David Stang, of the Virus Research Centre, International Computer Security Association.[73].

His studies have found that in a lot of cases, if a virus is resident when the IPX and NETX programs are loaded at the workstation, then that virus may be deactivated. This is mainly due to the way NETX "hooks" interrupts, which has the effect of denying some viruses access to the operating system features that they need in order to work. This does not always happen; certain viruses use different interrupts to the NETX program.

This research also found that many viral infections will disable NETX or IPX entirely, which results in the user of an infected machine being unable to access NetWare file servers at all. However, the use of virus scanning software at the workstation is still recommended, as some virus infections are still able to operate after a user has logged in.

ENSURING THE INTEGRITY OF DATA FILES

In the same way that program files need to be protected from unauthorised modifications, data files need to be protected so that the integrity of information in those files is preserved. If data is left unprotected, an attacker may be able to modify or destroy information.

Access controls

Access controls play a major part in protecting data integrity. NetWare administrators should ensure that users and groups have write access to only those directories where the data that they use resides. This may still represent a risk, as a malicious user with write access to a data file will be able to directly access that data outside of the application that is normally used to access it. This may result in corrupt or incorrect data being processed by other users of the application.

NetWare has no mechanisms that limit access to data files to certain programs. If a user can access data within a program, then the chances are that they will be able to access that data outside of a program. The only exception to this is where programs operate as client-server entities, where data access completely policed by the server process.

Cryptographic checksums (described above) may be used to verify the integrity of information files. NetWare has no built-in services of this type.

The Transaction tracking system (TTS)

NetWare does provide one mechanism that can be used to preserve the integrity of data files against corruptions caused by incomplete transactions. The Transaction Tracking System (TTS) ensures that modifications made to TTS data files are "atomic", meaning that these transactions are either completed in full, or have no effect on the data file.

Modifications to files that are marked as being tracked by TTS (with the Transactional attribute), are thus protected from corruptions that may be caused by a workstation crash, a loss of connection to the file server, or a file server crash.

SECTION SUMMARY

NetWare has several vulnerabilities that may lead to unauthorised modification of data and software. The most serious of these is that the SUPERVISOR and equivalent accounts have total access to the file server, regardless of rights and attributes settings. As a result, use of these accounts can constitute a major security risk if the workstation that the supervisory user is working from has been subverted by an intruder, or has been infected with a virus.

The execute-only flag may be used to prevent some kinds of unauthorised modifications, but its use is complicated by potential problems with application incompatibilities. Additionally, setting this attribute prevents the contents of the executable from being scanned for changes or from subversion by a companion type virus. Note that granting users modify or access control rights in directories where their access is otherwise restricted effectively cancels out those restrictions; the modify or access control rights can be used to change other rights and attributes settings in the directory.

Again, the complexity of NetWare access controls and the unpredictability of security equivalencies can play a major role in unauthorised modifications.

B.7 COMPROMISE OR MODIFICATION OF LAN TRAFFIC

Compromise of LAN traffic occurs when an unauthorised person gains access to data as it travels across the LAN medium. Modification of LAN traffic is the threat that messages, including the content or addressing of those messages, are modified by an attacker. The likelihood of these threats can be influenced by two factors: Encryption of LAN traffic, and physical protection of the LAN medium.

ENCRYPTION OF LAN TRAFFIC

While there may be extremely strict access controls at workstations and file servers, cleartext information that is transmitted over the network media may be compromised in a number of ways.

An attacker using a protocol analyser can decode the contents of data packets. This type of device does not need to be a separate item of hardware: Any workstation on a broadcast medium (such as Ethernet) that has a network interface card (NIC) capable of promiscuous mode is able to capture packets as they are transmitted across the LAN. A promiscuous mode NIC is able to listen in to *all* traffic on the network segment that it is attached to, rather than only traffic addressed to that particular workstation.

The main defence against data being compromised in this fashion is to encrypt data as it is carried across the network. Encryption can be carried out either by hardware or by software. Hardware encryption is typically faster, but is more expensive to implement as it requires expensive network interface cards. Software encryption, whilst cheaper than hardware solutions, is computationally intensive, and thus attracts performance penalties in terms of network response times at the server and workstation.

Currently, both hardware and software encryption systems are available for NetWare. However, encryption is *not* supplied as a standard feature of NetWare 3.11 or 3.12.

Hardware encryption solutions are available from several third party sources. Software encryption is available from Novell as phase two of their so-called two phase Security Enhancement Project . Phase two aims to encrypt *all* data that is transmitted across the LAN. (Phase one of the Security Enhancement Project deals with digital packet signatures, discussed in more detail in section B.8, Spoofing of LAN traffic).

Not only do encryption services reduce the risk that information is compromised as it travels across the LAN, it can also prevent data being modified. In order to modify a message, an attacker will have to decrypt the packet as it arrives, change the information contained in the message, re-encrypt the message (with the correct key) and then retransmit it.

The chances of this occurring in real-time are minimal. The nature of the Ethernet specification, which is very common in NetWare networks, is such that real-time modifications, even to unencrypted data, are very difficult to achieve without detection.

This may not be the case if the network has wide-area components, such as TCP/IP links between networks. Those links might be more easily compromised.

Modifications to LAN data can also be detected through the use of LAN Message Authentication Codes (MACs), which are a type of cryptographic checksum. NetWare currently has no cryptographical means of authenticating cleartext LAN message content.

PHYSICAL SECURITY OF NETWORK MEDIA

Encryption services can help reduce the risk of compromised or modified LAN data; physical security of the LAN medium can help to reduce the risk even further. By limiting who has physical access to LAN cables, the system administrator is able to reduce the risk of undetected tapping (vampire taps or EMR inductive taps) of the media.

Other solutions, including the use of intelligent hubs and wiring closets to route packets directly to their intended destination are also available. These systems, usually based on UTP (Unshielded Twisted Pair) Ethernet networks, act as a central clearing house for LAN packets. Each packet is passed onto its intended recipient as per normal, while other stations are passed a packet containing garbage information (this is necessary so that other network stations know that the network broadcast media is busy). Physical access to such hubs and wiring arrangements should be restricted.

SECTION SUMMARY

In summary, NetWare, in common with most other LAN systems that are based around broadcast media, is subject to a number of vulnerabilities that can result from the lack of standard encryption of network traffic, or from a lack of physical control of the network media.

Packet capture and media tapping are an accepted hazard of broadcast LAN systems. If sensitive or confidential data is stored on the network, then encryption should be used.

B.8 SPOOFING OF LAN TRAFFIC

Spoofing of LAN traffic involves the ability to receive a message by either masquerading as the legitimate receiving destination, or by masquerading as a legitimate sender and sending a false message to a target LAN device [7]. By masquerading as a legitimate sender, an attacker may be able to gain access to privileges and information that would not normally be granted (unauthorised access to LAN resources). By acting as a legitimate receiver, the

attacker may be able to gain access to information that they should not be privy to (compromise of LAN data).

To spoof the LAN, the attacker needs to convince the device being spoofed that the workstation they are mounting an attack from is a legitimate address of some other privileged victim user, for whom the request that they are making is normal.

This can be achieved in a variety of ways; the main prerequisite is that the victim is logged onto the network and is active during the data collection phase of the attack. The attacker might record a packet sequence that is part of a session between the victim and the LAN device, and then play that sequence back at a later time. The message may either be unmodified (playback attack), or may have a modified message content and workstation address (spoof attack). If no protection is in place, then the LAN device being spoofed will not know the difference between the legitimate user and the attacker.

The factors that can reduce this threat are:

- LAN traffic encryption.
- LAN traffic integrity controls.
- LAN message Non-repudiation services.

LAN TRAFFIC ENCRYPTION CONTROLS

One way to ensure that LAN devices are not spoofed is by implementing encryption schemes for LAN messages. These schemes, as discussed above, may be implemented in hardware or software, and may include encryption of certain parts of the message header, as well as the message itself. NetWare encryption (phase two) is designed to encrypt both header protocol information and message content.

Note that encryption will not prevent a playback attack, as encrypted packets may still be captured and subsequently retransmitted.

LAN TRAFFIC INTEGRITY CONTROLS

Another method to reduce the risk of a spoofing attack is to use MAC authentication of messages. As discussed on page 218, this provides a cryptographic assurance that message and message header data has not been tampered with. NetWare currently provides no security services to detect modification of message content, although once again, encryption services can limit this vulnerability.

NetWare does suffer from a major security vulnerability that is associated with a lack of integrity controls. Recently (late 1992) students at Lieden University in the Netherlands discovered a flaw in the way that NetWare handles NCP (Novell Core Protocol) requests. It is possible for a NetWare workstation posing as a more privileged client to send a forged NCP request to a NetWare server. With the right NCP request, the intruder is able to gain SUPERVISOR rights to the system, and consequently access to all network resources. [49].

The attack developed by these researchers takes advantage of NetWare's lack of authentication of NCP request messages. The only method that NetWare currently uses to check the authenticity of a user request is by examining the connection number that the request packet contains. If this connection number is changed by the attacker to that of another user (ie a SUPERVISOR user), then NetWare will process that request with the privileges of that user. A carefully crafted packet sent to a SUPERVISOR connection could, for example, request that the account that the attacker is using be granted SUPERVISOR equivalence.

Novell's response to this vulnerability was the NetWare Security Enhancement Project, which consisted of a two phase approach to strengthening NetWare security. Phase one addressed the NCP forgery vulnerability discovered at Leiden, while Phase two (discussed elsewhere in this report) addressed the problem of providing encryption services for LAN traffic.

Novell addressed the problem of NCP packet forgery by introducing a real-time NCP packet signature system. This system works by requiring the client and server to "sign" NCP packets. This signature, which changes for every packet transmitted, uses a cryptographically secure algorithm that uniquely identifies and verifies the sender of the NCP request. A number of verification combinations at both the client and server level are possible, ranging from no packet signatures being used, to a refusal on the part of the client or the server to complete a login if either do not provide the right level of packet signature.

The fix is implemented as a combination of new workstation shell software, an updated set of login and attach commands, and a NLM that runs on the server. The addition of packet signatures to a NetWare system carries a performance penalty of around three to ten per cent in terms of server and workstation CPU utilisation, and thus may not be feasible in environments where the file server is already under heavy loads.

Packet signature should be used in environments where sensitive information is stored at the server, or where a compromise of the SUPERVISOR account is unacceptable.

LAN MESSAGE NON-REPUDIATION SERVICES

Non-repudiation ensures that the parties in a communication cannot deny having participated in all or part of a communication [7]. Non-repudiation is usually achieved by using digital signatures as part of the communications protocol. These signatures provide proof that a sender sent a certain message, and that the sender is who they say they are. A variation on this arrangement is to have the receiver provide the sending station with proof of receipt.

NetWare does not provide non-repudiation services as a standard feature. However, phase one of the NetWare Security enhancement project, discussed above, can provide this service for NCP packets through its addition of a digital signature authentication mechanism to the NCP communications protocol.

SECTION SUMMARY

NetWare is vulnerable to spoofing attacks that take advantage of the NCP flaw described in this section. Although fixes are available to plug this hole, these may cause performance problems in LANs that are heavily utilised.

NetWare encryption services are not provided as standard, and if the Novell software solution is selected, this too may result in performance problems. NetWare only provides non-repudiation for NCP packets. Other types of communication may require third party protocols to be installed if non-repudiation is a requirement.

B.9 DISRUPTION OF LAN FUNCTIONALITY

A disruption of LAN functionality occurs when for any reason, the LAN is unavailable to users. This threat may be caused by physical disruptions, such as hardware problems or a system shutdown, or by logical disruptions, such as heavy network loads, or problems with the network operating system software.

Factors that influence unfettered LAN functionality are:

- Power availability and quality problems.
- Hardware Failures.
- Network Unavoidability.

- LAN device security.
- Network Operating System software problems.

POWER AVAILABILITY AND QUALITY PROBLEMS

Power cuts, brownouts and spikes can cause problems with the operation of the LAN. The most at risk device on the LAN is the file server. Because file servers may be servicing a number of users at once, any power fluctuations or outages are likely to affect large number of users, and an even larger number of files. Where possible, file servers should be connected to uninterruptable power supplies (UPS's) that provide enough time on battery power to shut the LAN down gracefully. Additionally, some UPS devices also provide power conditioning, which may be necessary in areas where the power is of low quality. NetWare supports the use of UPS devices for file servers.

HARDWARE FAILURES

Hardware failures can affect the availability and functionality of the network if they are not adequately prepared for. The most critical hardware problem of a LAN is when a file server hard drive fails. Other components of the file server such as memory, network interface cards or keyboards may be quickly replaced, but since server hard drives contain information, that data must be preserved and restored before the LAN can continue operating.

The obvious solution is to have replacement hardware ready and waiting for just such an occasion where the hard drive fails and availability of the LAN is critical. Usually, data and programs will have to be recovered from backups, although data may sometimes be recovered off a damaged hard drive. NetWare provides three options that can improve availability of NetWare file servers: Hot Fix, Disk Mirroring and Disk Duplexing.

Hot Fix

NetWare's Hot Fix feature reserves roughly two per cent of the space on each disk that it is activated for. After each disk write to a sector, NetWare reads the sector just written back off the disk and compares it with the same sector in memory. This read-after-write checking allows detection of defective areas of the disk. If the verification fails, then the sector is re-written to the reserved hot fix area, and the sector identified as defective is removed from use by the server.

In the event that verification of a sector written to the hot fix area of the disk fails, then NetWare will re-direct the sector to another part of the reserved hot fix area and try again.

The hot fix mechanism will attempt up to ten redirections before giving up and shutting down the hot fix mechanism for that disk. In this event, a console error message is generated informing the system administrator of the problem.

By monitoring the number of hot fixed areas, a system administrator can determine the relative health of a disk drive, and hopefully, can replace it before it fails completely.

Disk Mirroring

NetWare provides a feature known as disk mirroring. This works by using two separate physical disk drives, each of which has the same information written to it. If the read-after-write mechanism determines that a sector on one disk is faulty, that drive is shut down, and the other drive continues working. The inactive drive can then be removed for repair, without a loss of service.

Disk Duplexing

Disk duplexing involves installing two or more completely separate disk subsystems, including controllers and mirrored drives. In this case, redundancy is provided for defective drives, defective controllers, interfaces and power supplies. If any component in a disk channel fails, then the redundant channel takes over.

Disk duplexing has the added advantage of increased performance. File service requests can be fulfilled by whichever disk subsystem responds first, resulting in markedly improved performance.

NETWORK UNAVAILABILITY

Failure of the network media itself or problems with network capacity overloading fall into this area. The network may fail because of incorrect configuration of cables, the placement of cables too close to fluorescent lighting, faulty network terminators or cable or because of a faulty device on the LAN flooding the network with noise. Devices such as protocol analysers and cable analysers can sometimes be used to track the source of such problems down.

If network loads are too heavy, or if the LAN is unable to respond to changing traffic patterns, a loss of functionality may occur. NetWare provides the system administrator with information regarding file server utilisation and network loads. By monitoring this information, the administrator is able to plan upgrades to network hardware and perhaps plan reconfigurations of the LAN topology to remove bottlenecks and congested segments.

LAN DEVICE SECURITY

LAN devices should be secured from physical access as much as possible. As mentioned in section B.3 of this report (page 192), NetWare file servers in particular should be protected. Other items such as modems, gateways, routers, bridges etc should be protected from both unauthorised access and unintentional accidents, such as tripping over a cord.

NETWORK OPERATING SYSTEM SOFTWARE PROBLEMS.

A potential problem area that can lead to a disruption of LAN functionality is that of the network operating system itself. If files that are critical to the operation of the operating system become contaminated with viruses, or are corrupted due to hardware problems, LAN service may suffer. For this reason it is particularly important to retain backups not only of user data, but a set of backups of the file server software, including configurational files.

As mentioned at the start of this appendix, runaway or error-prone NLM applications may also disrupt the correct operation of the network operating system.

B.10 SUMMARY

This chapter has used taken a detailed look at the potential threats to NetWare 3.11 and 3.12 security, and the services that are provided to counter those threats. Within each threat area, a number of vulnerability factors were examined, and known problems with NetWare security were surfaced. As this report has discovered, NetWare does have a number of flaws that can make a good level of security difficult to achieve.

There is a general lack of auditing carried out by the system. Amongst other things, failed login attempts to non-existent accounts are not recorded; failed access to expired or time restricted accounts are not recorded; failed access to files or directories are not recorded. A NetWare administrator is not well equipped with information about how often or how severely attacks are made on system security.

The access control mechanisms implemented by NetWare are difficult to understand without a lot of practice, and are even more difficult to manage. Relatively minor changes to access rights at one level of a directory tree can affect major portions of that tree at lower subdirectory levels. NetWare rights and attributes do not apply in a uniform fashion to end users and supervisory users.

The NetWare security equivalence function, if improperly managed, can create major security vulnerabilities, due to the difficulty of determining its effects in some situations.

An intruder armed with a DOS disk editor who has physical access to an unprotected file server is able to surgically remove the entire identification and authentication mechanism from the system, effectively gaining complete control of the server. In addition, a "locked" file server can be unlocked by any print server operator who requests that a print server is shut down.

NetWare provides no encryption facilities, either for data stored at the server, or for data transmitted across the LAN. Additionally, NetWare is vulnerable to spoofing attacks that can give an intruder complete access to the file server. The only way to counter this vulnerability is to add NCP packet signature software, which may cause performance problems on heavily used servers.

There are many other potential problem areas that have been discussed in this report. A large part of NetWare security is dependant on the skills and attitude of the system administrator. In its favour, NetWare provides many more security features than some other PC LAN products, and a skilled administrator should be able to reduce NetWare vulnerabilities by recognising those problem areas and minimising system exposure to them.

NetWare system administrators require tools that provide more auditing information of security related system events, tools that automate the assignment of access controls so that a consistent access scheme can be maintained, tools that provide a more in depth analysis of potential security problems, tools that provide encryption of user files at the server and change management tools that assist in the control of changes to critical files and directories on the LAN.

APPENDIX C - NETWARE BINDERY OBJECTS

Object Class	Standard Properties	Type	
USER	IDENTIFICATION	Item	Contains up to 128 bytes of descriptive text about the user.
	GROUPS_I'M_IN	Set	Contains a list of user groups to which the user belongs.
	SECURITY_EQUALS	Set	Contains a list of bindery objects to which this user is security equivalent.
	LOGIN_CONTROL	Item	Contains structure defining access control settings and miscellaneous statistics related to system access.
	NODE_CONTROL	Item	Contains up to 10 network address masks from which this user is allowed to log in from.
	PASSWORD	Item	Contains the users encrypted password.
SUPERVISOR (as per user)	MANAGERS	Set	Set of users or groups that are authorised to act as workgroup managers.
	USER_DEFAULTS	Item	Default access controls for user LOGIN_CONTROL property.
USER_GROUP	IDENTIFICATION	Item	As above.
	GROUP_MEMBERS	Set	Contains a list of bindery objects which are members of the set.
PRINT_QUEUE	IDENTIFICATION	Item	As above.
	Q_USERS	Set	Users or groups of users authorised to place entries in the queue.
	Q_OPERATORS	Set	Users or groups of users authorised to act as queue operators.
	Q_DIRECTORY	Item	Server directory for queue data.
	Q_SERVERS	Set	Print Servers that service this queue.
PRINT_SERVER	IDENTIFICATION	Item	As above.
	PS_USERS	Set	Users or groups of users authorised to use print server.
	PS_OPERATORS	Set	Users or groups of users authorised to act as print server operators.
FILE_SERVER	OPERATORS	Set	Users or groups of users authorised to act as server operators.
	NET_ADDRESS	Item	Dynamic item property containing current internetwork address of file server.
	ACCT_LOCKOUT	Item	Current intrusion detection lockout settings.

Table C - 1 Sample bindery object types and their standard properties.

APPENDIX D
NETAUDIT USER INTERFACE PREVIEWS

This appendix previews NetAudit's user interface, and explains some of the interactive aspects of the package.

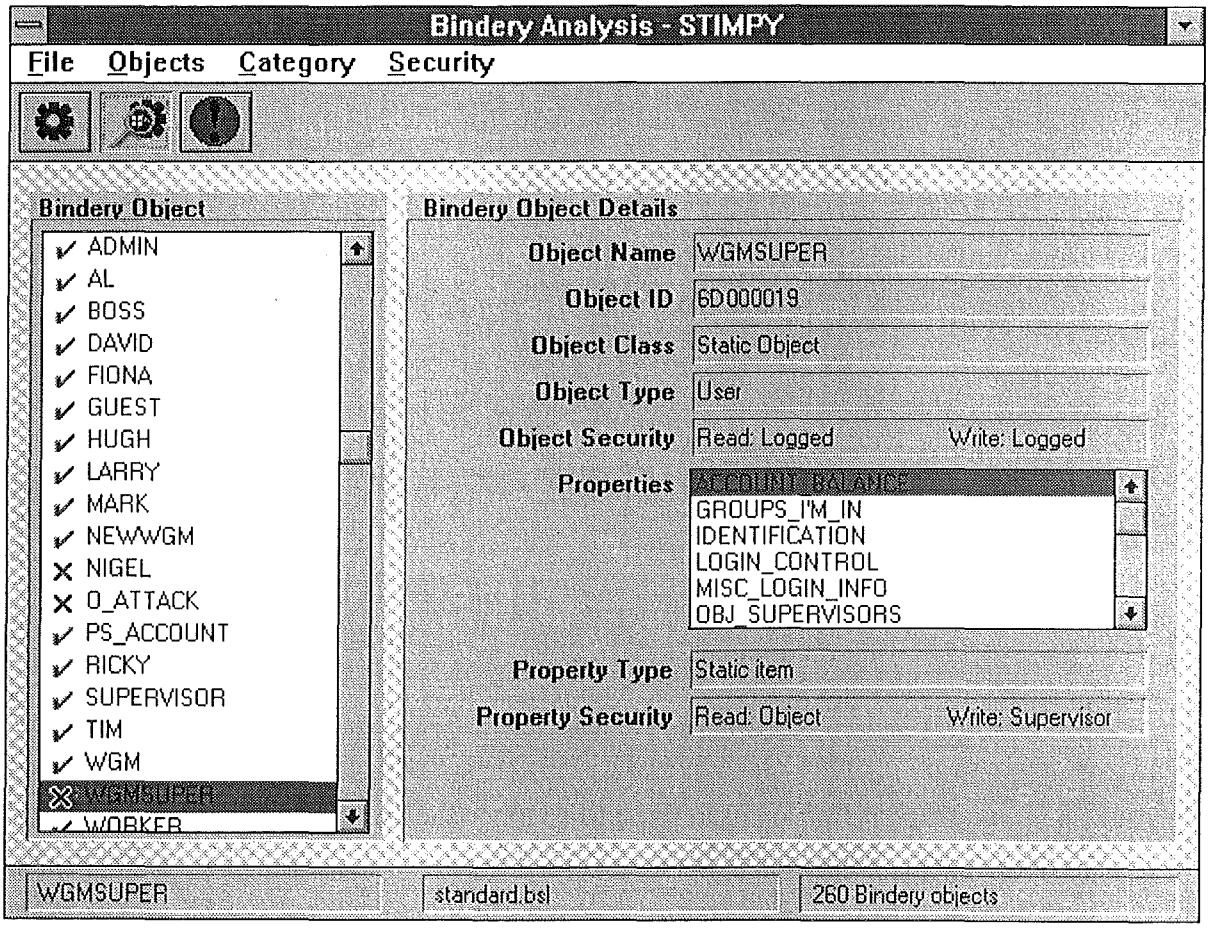


Figure D - 1 NetAudit bindery browser.

INTERACTIVE BINDERY BROWSER

The NetAudit Bindery Browser, as shown in Figure D - 1, allows the auditor to manually examine individual bindery objects. Objects may be selected on the basis of object type, and may be sorted according to either object type or object name. Using this browser, the auditor is able to examine the contents of item and set properties for any bindery object on

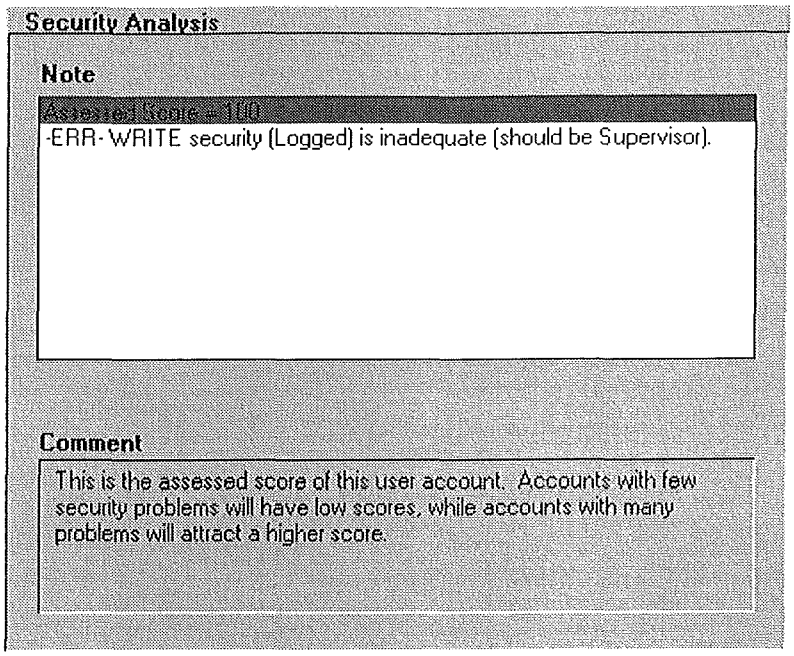


Figure D - 2 Bindery analysis window.

the target server. A list of objects referred to by set properties is available, and the contents of item properties is displayed in both text and binary forms.

Figure D - 2 shows the bindery security analysis subwindow of a failed object. The object that NetAudit has detected has an incorrectly specified object security setting, which allows any user (Logged) to make changes to its settings. The proper setting (Supervisor) is also shown.

INTERACTIVE USERBASE BROWSER

The userbase browser is shown in Figure D - 3 on the next page. This browser allows the auditor to interactively examine user objects that exist on a particular NetWare server. NetAudit permits the selection of users based on either the user account name, security equivalencies effective for the user, or based on the security assessment awarded to the account.

When a user is selected by the auditor, each category displays the current information for that account. Due to the large number of settings that affect user account security, these settings are divided into the display categories detailed below.

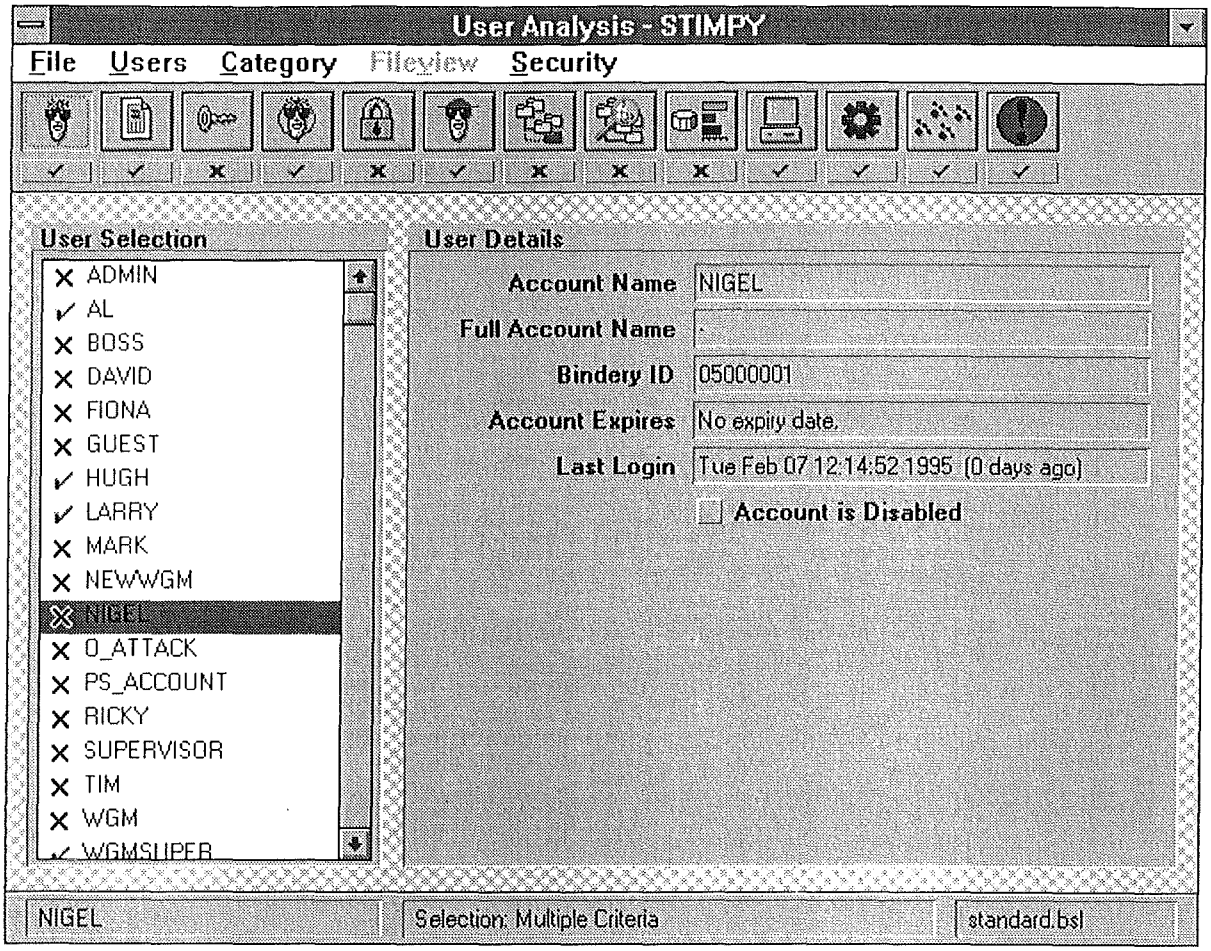


Figure D - 3 Userbase browser.

User Details.

This category includes the long name of the user, when the account was last used, the expiry date of the account, and the login state of the account.

Login Script.

This category allows the auditor to view user login scripts, as well as the creation date and file attributes of the login script file.

Password Settings.

This category displays user password security settings, including the minimum permissible length of the password, the password expiry date, whether unique passwords are required, and whether grace logins are allowed.

Account Restrictions.

This category displays the connection, node and time restrictions effective for the user.

Security Settings.

The Security Settings category indicates whether a user is SUPERVISOR equivalent, or is a Workgroup Manager. This category also displays the group membership of the user, as well as any user equivalencies that the account has.

Intrusion Detection Status.

This category displays the intrusion detection lockout status of the user account.

File System Rights.

This category indicates trustee rights that are assigned to this user. It includes trustee rights from direct trustee assignments, as well as rights from group memberships and security equivalencies.

File System View.

The file system view category allows the auditor to examine the real access that the user account has within the file system of the NetWare server that the account resides on. This view takes into account the effects of direct, group and security equivalence trustee assignments, as well as access rights inheritance from within the file system. Figure D - 4 shows this view.

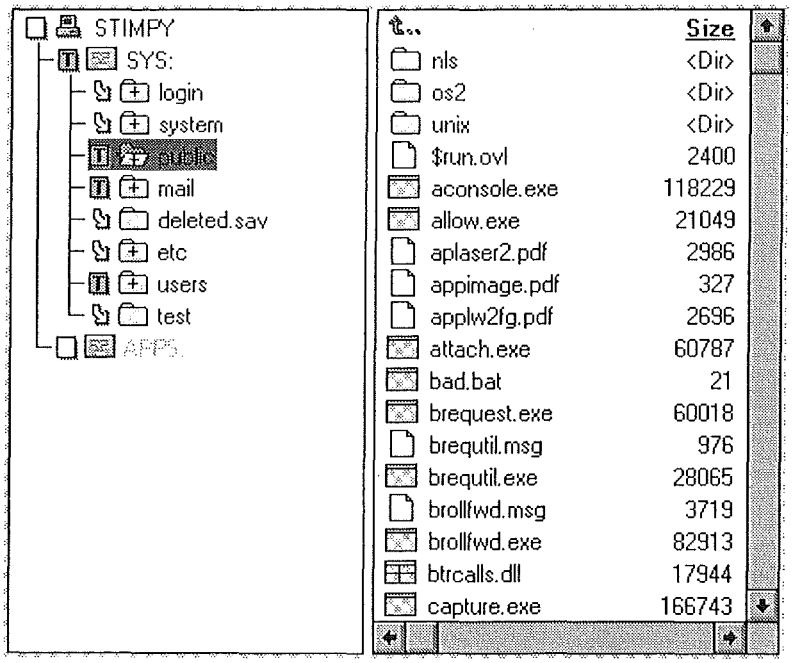


Figure D - 4 File system subview from within user browser. this view is showing the actual trustee access that the currently selected user has. Trustee assignments and inherited rights are shown as T boxes or arrows respectively .

Resource Usage.

This category displays the user account's consumption of file system space on each server volume.

Bindery Presence.

This category displays the bindery object associated with the user account, and allows the auditor to examine the bindery properties that belong to that user object.

Audit Trail List.

The Audit trail list shows any entries that exist in the system log that are associated with this user.

Security Analysis Window.

This window displays the outcome of the baseline analysis for this user. This view is shown in Figure D - 5.

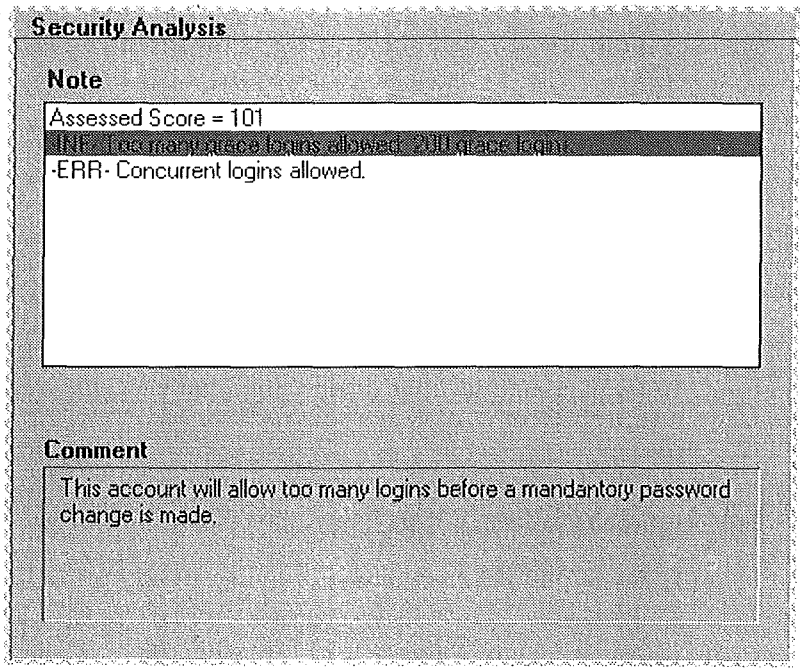


Figure D - 5 Security Analysis sub-window of the userbase browser showing the results of a detected errant user. Note the explanation given in the comment box below the listbox.

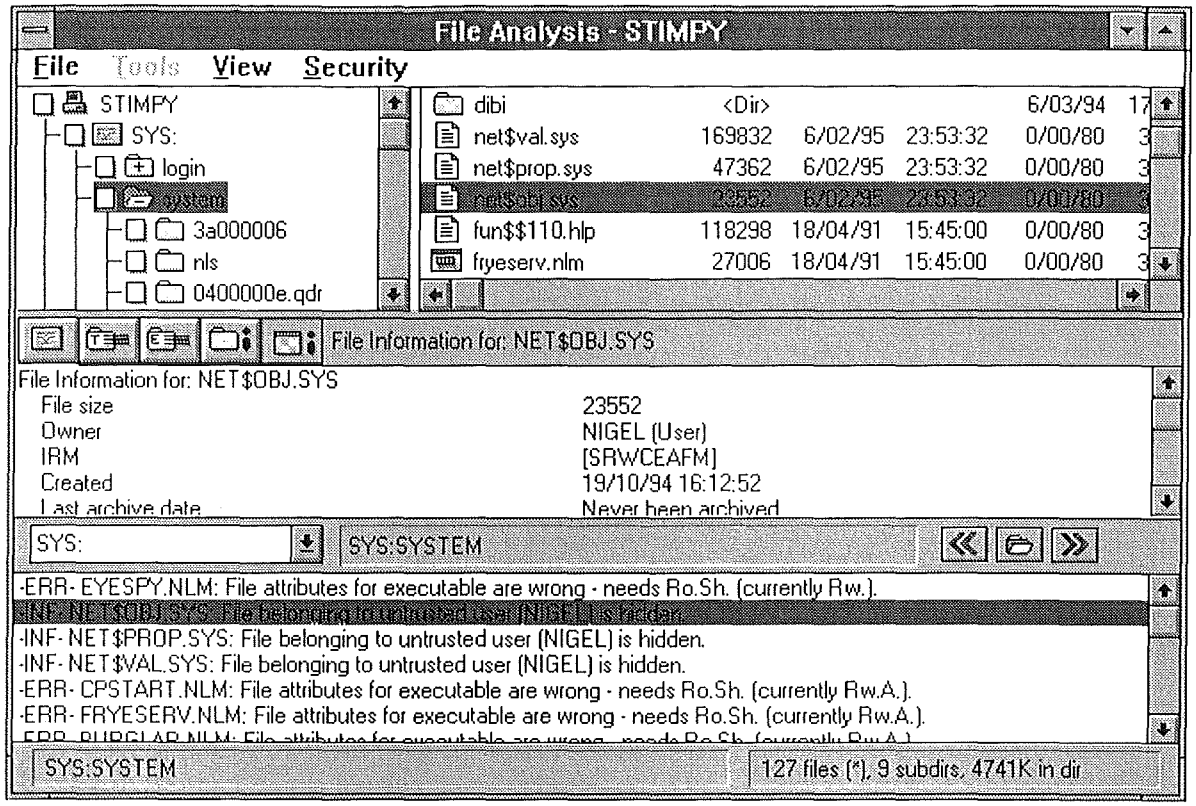


Figure D - 6 File system browser.

INTERACTIVE FILE SYSTEM BROWSER

The NetAudit interactive file system viewer is shown in Figure D - 6. The window is broken into three main panes. The top pane is a *directory and file browser*, in which the file system of the target server is displayed. Below this is located the *information* pane and toolbar that is used to display additional information about the currently selected directory or file. The bottom *results* pane and toolbar display the results of file system comparisons or baseline analysis runs.

NetAudit allows the auditor to select files to be displayed in the directory and file browser based on file wildcards, and supports sorting according to a number of criteria.

The information pane displays volume, directory or file information, depending upon the currently selected button in the toolbar immediately above it. From left to right, these toolbar buttons specify volume information, current directory trustees, current directory effective trustees, current directory information, and current file information.

This picture was taken just after a baseline run was performed, and as shown in the results pane, NetAudit has uncovered that the bindery files are hidden, and belong to a user that is not recognised as a supervisor of the system. This pane additionally shows several other executables (in this case NLM's) that have inappropriate attribute settings.

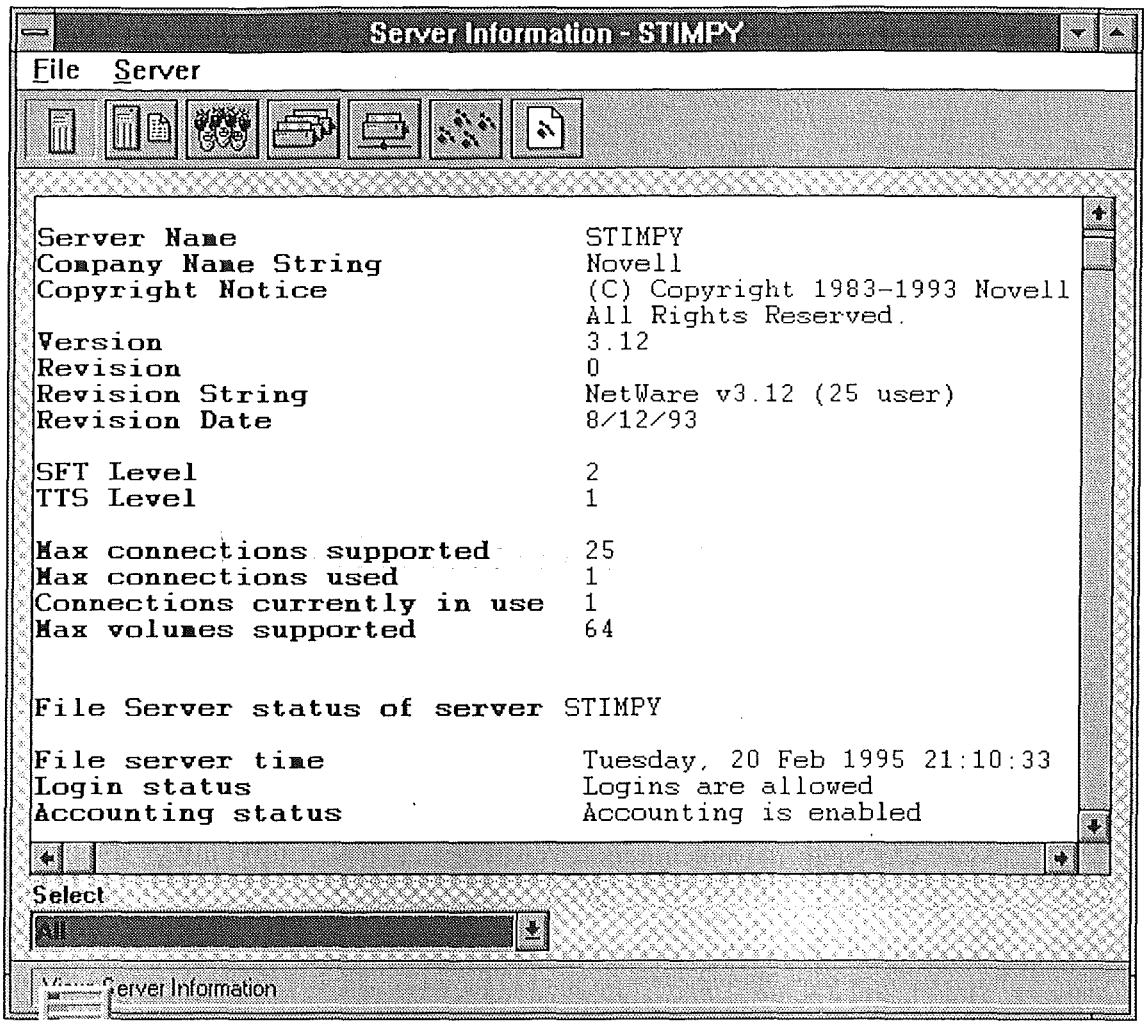


Figure D - 7 Server information browser

The three buttons to the right of the results toolbar allow the auditor to jump to the next or previous directory where baseline or comparison results have been flagged. The open folder button synchronises the file and directory browser display with the results display.

SERVER INFORMATION BROWSER

The server information browser is shown in Figure D - 7. This browser allows the auditor to examine a number of miscellaneous server configuration items, such as version information, group and device information, and log files.

REFERENCES

[1] Anderson T. E., Management Guidelines for PC Security, *SIGSMALL/PC Notes Vol 18, Nos. 1 & 2,* Spring/Summer 1992.

[2] Anthes G. H., Poll finds security less than passable, *ComputerWorld (USA),* April 18, 1994, p.63.

[3] Baldwin R. W., Kuang: Rule-Based Security Checking, Technical Report included in COPS 1.04 distribution, MIT Computer Science Lab (Programming Systems Research Group), circa 1990.

[4] Bellovin S., There Be Dragons, *Proceedings of the Third USENIX Unix Security Symposium*, Baltimore, September 1992.

[5] Bontchev V., Possible Virus Attacks Against Integrity Checkers And How To Prevent Them, Technical report, Virus Test Center, University of Hamburg, 1993.

[6] Buck E. R., Introduction to Data Security and Controls (2nd Ed.), QED Technical Publishing Group, 1991.

[7] Carnahan L., A Local Area Network Architecture, *Proceedings 15th National Computer Security Conference,* 1992.

[8] Carroll J. M., Computer Security (2nd Ed.), Butterworths, 1987.

[9] Chang S., Priorities for LAN security - A case study of a federal agency's LAN security, *Proceedings 15th National Security Conference,* 1992.

[10] Cheswick W. R., An Evening with Berferd, *Proceedings of the Winter USENIX Conference*, San Francisco, January 1992.

[11] Chorley B. J., Price W.L., Security assessment and conformance testing, *Proceedings of the IFIP TC11 Seventh International Conference on Information Security: Creating Confidence in Information Processing*, North-Holland, 1991, pp.43-54.

[12] Clarke D. J., Novell's CNA Study Guide, Novell Press (Sybex), 1993.

[13] Cooper J. A., Computer & Communications Security - Strategies for the 1990s, McGraw-Hill Commications Series, 1989.

[14] Cohen F., Some initial results from the QUT Virus Research Network (draft), As yet unpublished, 1993.

[15] CIAC, Automated attacks on networked computers, *CIAC advisory bulletin D-25*, CIAC (The Computer Incident Advisory Capability), September 1993.

[16] DataPro, Cental Federal Systems, Inc. Net/Assure, *DataPro Reports on Information Security,* January 1992.

[17] DataPro, Pyramid Development Corp. Net/DACS, *DataPro Reports on Information Security,* January 1992.

[18] David J., LAN Security Standards, *Computers and Security No. 11 (1992)* 1992.

[19] Denning P. J. (ed), Computers Under Attack - Intruders, Worms and Viruses, Addison-Wesley, 1990.

[20] Department of Defence Trusted Computer System Evaluation Criteria (The Orange Book, or TCSEC), US Department of Defence Standard 5200.28-STD, December 1985.

[21] Eastwood A., "Novell to improve NetWare security (Novell Inc. discovers hackers can penetrate NetWare LANs, to address problem)", *Computing Canada, Vol. 18 Iss. 23,* November 1992.

[22] Farmer D., Cops and Robbers - U*NX System Security, from COPS 1.04 distribution, circa 1989.

[23] Farmer D., Spafford E., The COPS Security Checker System, *Proceedings of the Summer USENIX Conference,* (also available as Purdue University Technical Report CSD-TR-993), 1990.

[24] Farquhar C., The Development of an Automated Security Review Methodology for New Zealand Government Computer Systems, Christchurch, University of Canterbury, 1993 (Thesis: M.Com).

[25] Federal Criteria for Information Technology Security -Volume 1 (draft version 1.0), National Intstitute of Standards and Technology (NIST), Gaithersburg, December 1992.

[26] Garfinkle S., Spafford E., Practical Unix Security, O'Reilly & Associates, 1991.

[27] Gilbert I. E., Guide for Selecting Automated Risk Analysis Tools, National Institute of Standards and Technology (NIST) SP-500-174, Gaithersburg.

[28] Goldis P. D., Questions and answers about tiger teams, *The EDP Audit, Control, and Security Newsletter Vol. XVII, No. 4*, October 1989. pp.1-10.

[29] Greenwald J., Breaking into your own system, *Business Insurance Vol. 28 Iss. 23*, June 6 1994. p.11.

[30] Gupta S., Gligor V. D., Experience with a Penetration Analysis Method and Tool, *Proceedings 15th National Security Conference*, 1992.

[31] Harmonised Information Technology Security Evaluation Criteria (ITSEC), version 1.2, UK Department of Trade and Industry, London, June 1991.

[32] Holbrook P., Reynolds J. (eds), Site Security Handbook (rfc1244), Internet Engineering Task Force, July 1991.

[33] Johnston M., Vasiliki S., Testing a Secure Operating System, *Proceedings of the 13th National Computer Security Conference*, 1990.

[34] Kim G. H., Spafford E. H., The Design and Implementation of Tripwire: A File System Integrity Checker, Purdue University Technical Report CSD-TR-93-071, November 19 1993.

[35] Kim G. H., Spafford E. H., Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection, Purdue University Technical Report CSD-TR-93-071, February 21 1994.

[36] Kim G. H., Spafford E. H., Readme for Tripwire, included in Tripwire (version 1.1) distribution, December 1993.

[37] Klaus C., Internet Security Scanner v1.21, Documentation included in ISS 1.21 distribution, 1993.

[38] Linde R., Operating system penetration, *Proceedings of the National Computer Conference Vol. 44*, AFIPS Press,1975, pp.361-368.

[39] Lamb J., Jarocki S. R., Seijas, A. M., NetWare Security: Configuring and Auditing a Trusted Environment, Novell Cooperative Research Report, Novell Inc, 1991.

[40] Lunt T. F., A survey of intrusion detection techniques, *Computers & Security 12*, 1993, pp.405-418.

[41] Muffett A., Almost Everything You Ever Wanted To Know About Security (but were afraid to ask!), Internet comp.security.misc newsgroup FAQ, 22 October 1993.

[42] Muffet A., Crack Version 4.0A - A Sensible Password Checker for Unix, University College of Wales, included in Crack 4.0A documentation.

[43] Mungo P., Clough B., Approaching Zero, Random House Publishing, 1992.

[44] Myers P. A., Subversion: The Neglected Aspect of Computer Security, Naval Postgraduate School, Monterey California, June 1980 (Msc Thesis).

[45] Nance B., Thompson T., Smith B., Automatic NetWare Log-Ins, *BYTE Magazine*, March 93.

[46] Krohn N., "Novell releases NetWare server security patch", *PC Week V9 issue 40.* October 1992.

[47] Novell Inc., Netware System Interface Technical Overview, Addison-Wesley, 1990

[48] Novell Inc., NetWare 3.11 Concepts Manual, March 1991.

[49] Novell Inc., NCP Packet Signature for NetWare 3.11 - Patch Reference Document, Security patch documentation released by Novell to combat HACK.EXE, 1992/1993.

[50] Neumann P. G., Computer system security evaluation, *Proceedings of the National Computer Conference Vol. 47*, AFIPS Press,1978, pp.1087-1095.

[51] Neumann P. G., Parker, D. B., A Summary of Computer Misuse Techniques, *Proceedings of the 12th National Computer Security Conference*, October 1989.

[52] NZSIT 100: The Security of Computer Systems, The New Zealand Government Communications Security Bureau, April 1992.

[53] Pfleeger C., Pfleeger S., Theofanos M., A Methodology For Penetration Testing, *Computers & Security* 8, 1989 pp.613-620.

[54] Pfleeger S., Measuring Software Reliability, *IEEE Spectrum Vol. 29 Iss. 8*, August 1992, pp.56-60.

[55] Polk W. T., Automated Tools for Testing Computer Systems Vulnerability, National Institute of Standards and Technology (NIST) SP-800-6, December 1992.

[56] Risks Forum, "Hospital Admits Errors In Treatment Of Cancer Patients" (requoted article), *Internet Risks Forum Vol. 13 Iss. 12,* 7 Febrary 1992.

[57] Risks Forum, "Cancer Blunder Will Cost Health Authority Millions" (requoted article), *Internet Risks Forum Vol. 15 Iss. 5,* 30 September 1993.

[58] Risks Forum, "Pentagon computers cracked" (requoted article), *Internet Risks Forum Vol. 16 Iss. 27*, 21 July 1994.

[59] Risks Forum, "Pentium FDIV Bug" (various authors), *Internet Risks Forum, Vol 16. Iss 59*, 30 November 1994.

[60] Risks Forum, "CapAccess compromised" (requoted article), *Internet Risks Forum Vol 16. Iss. 64*, 24 February 1995.

[61] Rochlis J. A., Eichin M. W., With Microscope and Tweezers: The Worm from MIT's Perspective, *Communications of the ACM Vol. 32 No. 6*, June 1989, pp. 689-698.

[62] Rothke B., "SmartPass NLM makes converts out of end-users with insecure passwords", *Infoworld, Vol. 16, Iss. 20*, May 16 1994, p.49.

[63] Russell D., Gangemi G. T., Computer Security Basics, O'Reilly & Associates, 1991.

[64] Ruthberg *et al.*, Guide to Auditing for Controls and Security - A System Development Life Cycle Approach, National Bureau of Standards (now NIST) Special Publication 500-153, April 1988.

[65] Safford D. R., Schales D. L., Hess D. K., The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment, *Proceedings of the Fourth USENIX Security Symposium*, 1993.

[66] Salamone S., A Magic Bullet for NetWare Viruses, *Data Communications Magazine,* December 1992.

[67] Salamone S., Internetwork Security: Unsafe at Any Node?, *Data Communications Magazine,* September 1993.

[68] Scherlis W., Squires S., Pethia R., Computer Emergency Response, (Appears in [19]), 1990.

[69] Schultz E., Pethia R. Dalton J. Computer Emergency Respons Teams: Lessons Learned, *Proceedings of the 13th National Computer Security Conference*, 1990.

[70] Seyfert J., Hoelscher B., Understanding Local Area Network Security, *The EDP Audit, Control, and Security Newsletter,* June 1992.

[71] Sheldon T., Novell Netware 386 - The Complete Reference, McGraw-Hill, 1990.

[72] Spafford E. H., Crises and Aftermath, *Communications of the ACM Vol. 32 No. 65*, June 1989, pp.678-687. (Reprinted in [19]).

[73] Stang D. J., Virus Dangers to NetWare LAN's - Fact vs. Fiction, *NetWare Users International - NetWare Connection,* Jan/Feb 1993.

[74] Steffora A., Cheek M., "Hacking Goes Legit", *Industry Week Vol 243 Iss. 3*, 7 February 1994, pp.43-46.

[75] Stoll C., The Cuckoo's Egg, DoubleDay, 1989.

[76] Thompson K., Reflections on trusting trust, *Communications of the ACM Vol. 27 No. 8*, August 1984 pp.761-763. (Reprinted in [19])

[77] Wack J. P., Carnahan, L. J., Virus Prevention for Personal Computers and Associated Networks, NIST Special Publication 500-166, National Institute of Standards and Technology, Gaithersburg, August 1989.

[78] Wallis A., "Security Cover Needs To Include Administrator", *The Dominion InfoTech Weekly*, New Zealand, March 29 1993.

[79] Watson N. G., A Review of Current Intrusion-Detection Systems and Methodologies, Interim technical report for the New Zealand GCSB, University of Canterbury, March 1993.

[80] Watson N. G., A Vulnerability Analysis of Novell NetWare 3.11, *Interim technical report for the New Zealand GCSB,* July 1993.

[81] Watson N. G., NetAudit: An Automated tool for assessing NetWare 3.11 and 3.12 Security, *Interim technical report for the New Zealand GCSB,* March 1994.